МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: **«Проєктування та розробка вебзастосунку для управління фітнес-клубом з системою індивідуальних і групових тренувань та абонементів»**

Виконав: студент 4 курсу, групи КН-4__
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Носко Тарас Володимирович

Керівник: *викладач кафедри інформаційних технологій та аналітики даних Місай Володимир Віталійович*

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики Донецького національного університету імені Василя Стуса Загоруйко Любов Василівна*

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри інформаційних технологій та аналітики даних _____ (проф., д.е.н. Кривицька О.Р.)
Протокол № 11 від « 20 » травня 2025 р.

Острог – 2026 р.

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Проєктування та розробка вебзастосунку для управління фітнес-клубом з системою індивідуальних і групових тренувань та абонементів

Автор: Носко Тарас Володимироч

Науковий керівник: викладач кафедри ІТАД, Місай Володимир Віталійович
Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: ____ *(кількість сторінок роботи) с., ____ (кількість рисунків) рис., _____ (кількість таблиць) табл., _____ (кількість додатків) додатків, _____ (кількість джерел) джерел.*

Ключові слова: *дизайн, вебдодаток, спортзал, UX/UI, прототипування, користувацький інтерфейс, запис на тренування, календар тренувань, тренери, адаптивний дизайн.*

Пояснювальна записка до кваліфікаційної роботи: *69 с., 14 рис., 1 табл., 3 додатки, 40 джерел.*
Ключові слова: *вебзастосунок, фітнес-клуб, управління тренуваннями, індивідуальні тренування, групові тренування, онлайн-розклад, бронювання тренувань, абонементи, заявки до тренера, спеціалізації тренерів, рольова модель доступу, ASP.NET Core, Entity Framework Core, PostgreSQL, React, Tailwind CSS, Vite, JWT-автентифікація, REST API.*
Короткий зміст праці:
У кваліфікаційній роботі розроблено сучасну інформаційну систему для управління фітнес-клубом з підтримкою каталогу видів тренувань, гнучкої системи бронювання індивідуальних та групових занять, ведення абонементів і взаємодії між відвідувачами, тренерами й адміністрацією. Особливу увагу приділено покращенню доступності та зручності користування завдяки адаптивному інтерфейсу, інтерактивному календарю розкладу й чітко розмежованим рольовим кабінетам, що дозволяє кожній категорії користувачів швидко виконувати потрібні дії без зайвих кроків.
Застосунок розроблено на основі ASP.NET Core із використанням Entity Framework Core та бази даних PostgreSQL, а клієнтську частину реалізовано на React (Vite, Tailwind CSS). Він забезпечує фільтрацію тренувань за категоріями (тренажерний зал, стрейчинг, TRX, пілатес, сайклінг), за форматом (групове / індивідуальне / самостійне) та за спеціалізаціями тренерів. У систему інтегровано перевірку зайнятості тренера у визначені робочі години (Пн–Пт, 09:00–17:00), обмеження

кількості учасників у груповому занятті, JWT-автентифікацію та рольовий контроль доступу для адміністратора, тренера й відвідувача. Також реалізовано систему заявок до тренера з підтвердженням або відхиленням і автоматичними сповіщеннями про результат, ведення абонементів з контролем залишку тренувань та неможливістю одночасно мати два активні абонементи, історію тренувань відвідувача, фільтри «Майбутні / Минулі / Групові / Індивідуальні», а також можливість самостійного тренування для клієнтів з активним абонементом. Розроблена система поєднує гнучку модульну архітектуру (Domain / Application / Infrastructure / API), зручність використання, прозору бізнес-логіку обліку відвідувань та сучасні підходи у сфері веброзробки, що робить її масштабованим рішенням для автоматизації роботи фітнес-клубу.

_____

ANNOTATION

of qualification paper
for bachelor's degree

Theme: Design and development of a web application for managing a fitness club with a system of individual and group training and subscriptions

Author: Nosko Taras

Scientific supervisor: Lecturer of the Department of ITAD, Misai Volodymyr Vitaliiovych

Defended: «.....»..... 2026 year.

Explanatory note to the qualification thesis: *69 pages, 14 figures, 1 tables, 3 appendices, 40 references.*

Keywords: *web application, fitness club management, training scheduling, individual training, group training, online schedule, booking system, training requests, membership subscriptions, trainer specializations, role-based access control, ASP.NET Core, Entity Framework Core, PostgreSQL, React, Tailwind CSS, Vite, JWT authentication, REST API.*

Abstract:
This bachelor's thesis presents the development of a modern information system for fitness club management that supports a catalog of training types, a flexible booking system for individual and group sessions, membership administration, and structured interaction between visitors, trainers, and administrative staff. Special attention is given to improving accessibility and user convenience through a responsive interface, an interactive scheduling calendar, and clearly separated role-based dashboards that allow each category of users to perform the required actions quickly and without unnecessary steps.
The application is developed using ASP.NET Core with Entity Framework Core and a PostgreSQL database, while the client side is implemented in React (Vite, Tailwind CSS). It provides filtering of trainings by category (gym, stretching, TRX, Pilates, cycling), by format (group / individual / self-workout), and by trainer specializations. The system integrates trainer availability checks within defined working hours (Mon–Fri, 09:00–17:00), participant-count limits for group sessions, JWT authentication, and role-based access control for the administrator, trainer, and visitor.
Additionally, the system implements a trainer-request workflow with confirmation or rejection and automatic notifications about the outcome, membership administration with session-balance tracking and a constraint preventing two simultaneously active subscriptions, a visitor training history, filters such as "Upcoming / Past / Group / Individual", and a self-workout option for clients holding an active membership.
The developed system combines a flexible modular architecture (Domain / Application / Infrastructure / API), ease of use, transparent business logic for visit accounting, and state-of-the-art approaches in modern web development, making it a scalable solution for

automating the operation of a fitness club.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

API - Application Programming Interface: набір правил і методів, за допомогою яких окремі компоненти програм можуть взаємодіяти між собою.

ASP.NET Core - кросплатформений вебфреймворк від Microsoft для створення високопродуктивних вебзастосунків і RESTful-сервісів.

CRUD - Create, Read, Update, Delete: чотири основні операції при роботі з інформацією в базах даних.

CSS - Cascading Style Sheets: мова опису зовнішнього вигляду документа, написаного мовою розмітки.

DDD - Domain-Driven Design: підхід до проєктування ПЗ, у якому ядром системи є предметна область та її модель.

DI - Dependency Injection: шаблон проєктування, за якого залежності між компонентами впроваджуються ззовні, що спрощує тестування та підтримку коду.

DOM - Document Object Model: модель, що представляє HTML- або XML-документ у вигляді дерева об’єктів, з якими можна взаємодіяти через скрипти.

DTO - Data Transfer Object: об’єкт для передачі даних між шарами застосунку або між клієнтом і сервером.

EF Core - Entity Framework Core: об’єктно-реляційний фреймворк (ORM) для .NET для роботи з реляційними БД через C#-об’єкти.

HTML - HyperText Markup Language: стандартизована мова розмітки для створення вебсторінок.

HTTP / HTTPS - HyperText Transfer Protocol (Secure): протокол обміну даними між клієнтом і сервером у вебі; HTTPS — захищена версія з шифруванням TLS.

ID - Identifier: унікальне цифрове або символьне позначення для ідентифікації елемента в системі.

JS - JavaScript: мова програмування, переважно для інтерактивної поведінки на стороні клієнта у вебзастосунках.

JSON - JavaScript Object Notation: текстовий формат обміну структурованими даними між клієнтом і сервером.

JWT - JSON Web Token: стандарт компактного токену для безпечної передачі тверджень (claims) між сторонами; використовується для автентифікації та авторизації.

LINQ - Language Integrated Query: мова запитів, інтегрована в C#, для запитів до колекцій і баз даних у вигляді коду.

MVC - Model-View-Controller: архітектурний шаблон, що розділяє застосунок на модель даних, представлення та контролер.

.NET - універсальна платформа від Microsoft для розробки ПЗ на різні операційні системи та пристрої.

ORM - Object-Relational Mapping: взаємодія з БД через об’єкти коду, що відповідають таблицям.

PostgreSQL - СУБД з відкритим кодом, що підтримує складні запити та об’єктно-реляційний підхід.

REST - Representational State Transfer: архітектурний стиль для вебсервісів: ресурси через URL, операції стандартними HTTP-методами.

RBAC - Role-Based Access Control: розмежування доступу за ролями користувачів (наприклад, Адміністратор, Тренер, Відвідувач).

SPA - Single-Page Application: вебзастосунок, що завантажується одним HTML-документом і динамічно оновлює вміст без повного перезавантаження сторінки.

UI - User Interface: користувацький інтерфейс взаємодії з системою.

UML - Unified Modeling Language: стандартизована мова діаграм структури та поведінки програмних систем.

URL - Uniform Resource Locator: уніфікований ідентифікатор ресурсу в мережі (розташування та спосіб доступу).

UX - User Experience: досвід користувача під час взаємодії з системою (зручність, ефективність, сприйняття).

Vite - інструмент збирання та розробки фронтенд-застосунків із швидким dev-сервером і оптимізованою збіркою.

БД - база даних: структурований набір інформації, що зберігається та керується за допомогою СУБД.

СУБД - система управління базами даних: ПЗ для створення, обслуговування та взаємодії з базами даних.

Зміст

ВСТУП

У наш час цифровізація сфери послуг набуває дедалі більшої ваги, а фітнес-індустрія активно впроваджує онлайн-інструменти для планування занять, обліку відвідувань і взаємодії між клієнтами, тренерами та адміністрацією. Зростає кількість людей, які очікують від фітнес-клубу не лише якісних тренувань, а й зручного доступу до розкладу, можливості заздалегідь забронювати місце на групове заняття, узгодити індивідуальний час із тренером та прозоро керувати абонементом без черг і паперових журналів. Водночас з'являються типові труднощі: перевантажений розклад, ризик подвійного бронювання, неузгодженість між «заявкою», «підтвердженням» і «списанням» відвідування з абонемента, а також потреба в єдиному інформаційному просторі для різних ролей користувачів.

Розробка вебзастосунку для управління фітнес-клубом із підтримкою каталогу видів тренувань, бронювання індивідуальних і групових занять, обліку абонементів і заявок до тренера дозволяє зробити процес організації тренувань більш передбачуваним, керованим і зручним як для відвідувачів, так і для персоналу. Такий підхід не лише зменшує адміністративне навантаження, а й підвищує прозорість правил запису, обмежень за кількістю учасників у групі та контролю доступних часових вікон тренера.

Ринок фітнес-послуг характеризується високою конкуренцією та зростаючими вимогами до сервісу. Для відвідувача важливо не просто «записатися на тренування», а впевнено обрати формат заняття (групове, індивідуальне або самостійне), відповідну категорію та тренера з потрібними спеціалізаціями, враховуючи реальну зайнятість і наявність активного абонемента. Для тренера та адміністратора критично мати інструменти керування розкладом, користувачами й тарифними планами в межах єдиної системи з розмежуванням доступу.

Мета кваліфікаційної роботи - спроєктувати та реалізувати сучасний вебзастосунок для управління фітнес-клубом, який поєднує серверну частину на платформі .NET, реляційну базу даних PostgreSQL та клієнтський інтерфейс на

React, забезпечуючи надійну автентифікацію, авторизацію за ролями та зручну взаємодію користувачів із розкладом і абонементами.

Для досягнення мети сформульовано такі **завдання**:

- Зібрати та проаналізувати сучасні підходи до створення вебзастосунків для сфери фітнес-послуг і типові вимоги до обліку відвідувань та абонементів;
- Спроєктувати предметну область і модель даних для користувачів, тренувань, бронювань, групових сесій, абонементів і заявок до тренера;
- Реалізувати серверну логіку REST API на ASP.NET Core з використанням Entity Framework Core та міграцій схеми бази даних;
- Забезпечити автентифікацію на основі JWT і рольову модель доступу для адміністратора, тренера та відвідувача;
- Розробити клієнтський інтерфейс на React із адаптивним дизайном, календарем розкладу, фільтрами за типом занять і спеціалізаціями тренерів;
- Перевірити коректність ключових сценаріїв: бронювання, обмеження групових місць, перевірка зайнятості тренера, купівля/ведення абонемента, заявки до тренера та сповіщення про результати взаємодії.

Об’єкт дослідження - вебзастосунок для управління фітнес-клубом із підтримкою індивідуальних і групових тренувань та системою абонементів.

Предмет дослідження - методи та засоби реалізації серверної бізнес-логіки, моделі даних, REST API, автентифікації та авторизації, а також підходи до побудови клієнтського інтерфейсу для планування тренувань і адміністрування контенту клубу.

Отже, створення такого вебзастосунку дозволяє підвищити ефективність організаційних процесів у фітнес-клубі, зменшити ймовірність помилок при ручному обліку та забезпечити сучасний рівень взаємодії з користувачем через єдину цифрову платформу. Це розширює можливості відвідувачів щодо планування

занять і робить керування розкладом та абонементами більш прозорим і керованим для персоналу клубу.

РОЗДІЛ 1

ТЕОРЕТИЧНІ АСПЕКТИ ПРОЄКТУВАННЯ ТА РОЗРОБКИ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ФІТНЕС-КЛУБОМ

1.1. Опис предметного середовища вебзастосунку для фітнес-клубу

Сфера фітнес-послуг дедалі активніше переходить у цифровий формат, а власні онлайн-сервіси стають одним із основних інструментів утримання клієнтів для компаній, що працюють у галузі здоров'я та активного способу життя. Цифрові розклади, онлайн-бронювання, особисті кабінети тренерів і відвідувачів дозволяють охопити ширшу аудиторію та надати клієнтам зручність у плануванні занять, особливо за допомогою персоналізованих інструментів, які адаптуються під розклад конкретного користувача.

Вебзастосунок для управління фітнес-клубом — це програмне рішення, що спеціалізується на автоматизації процесів запису, обліку та проведення тренувань: індивідуальних, групових і самостійних. Це сучасний інструмент цифрового управління, який дозволяє користувачам обирати тренера, записуватися на заняття, керувати своїм абонементом і взаємодіяти з адміністрацією клубу через інтернет.

Предметним середовищем у даному випадку слугує ринок онлайн-сервісів для фітнес-індустрії. Ключовими учасниками процесу є: відвідувачі (клієнти клубу), тренери, адміністратори, інформаційна система клубу, база даних користувачів і тренувань, модуль розкладу, модуль абонементів та модуль заявок до тренера. Автоматизації підлягає процес планування та проведення тренувань з урахуванням індивідуального графіка клієнта, спеціалізацій тренерів і обмежень за кількістю учасників у груповому занятті. Планується створити вебзастосунок, який дає можливість не тільки записатися на тренування, а й полегшує цей процес шляхом фільтрації за категоріями та форматами занять, інтерактивного календаря розкладу й рольових кабінетів для різних типів користувачів.

За даними галузевих оглядів, фітнес-індустрія України поступово відновлює докризові обсяги: близько 7–8 % дорослого населення регулярно відвідує клуби, а більшість сучасних мереж активно впроваджують онлайн-сервіси для запису, обліку

відвідувань і комунікації з клієнтами. У глобальному масштабі ринок цифрових сервісів для фітнесу (онлайн-розклади, мобільні кабінети, системи бронювання) демонструє стабільне зростання понад 15 % на рік, що свідчить про актуальність тематики розробки.

Згідно з аналітичними звітами, серед основних причин, через які клієнти обирають один клуб замість іншого, понад 60 % опитаних називають саме зручність онлайн-запису та прозорість роботи з абонементом, тоді як ціна стоїть лише на другому місці . Це підтверджує, що цифрова складова сервісу безпосередньо впливає на лояльність клієнтів і конкурентоспроможність клубу.

Таким чином, можна зазначити, що предметне середовище вебзастосунку для управління фітнес-клубом є комплексною системою, яка поєднує різноманітні цифрові інструменти: керування користувачами, графіками тренерів, розкладом групових і самостійних занять, а також обліком абонементів. Успішне функціонування застосунку залежить не лише від дизайну або обсягу функціональності, а й від здатності забезпечити узгоджену та зручну взаємодію між адміністратором, тренером і відвідувачем на кожному етапі організації тренування.

1.2. Огляд наявних аналогів вебзастосунків для управління фітнес-клубом

Для порівняння було обрано два популярні сервіси, які використовуються українськими фітнес-клубами та студіями: **Mindbody** (mindbodyonline.com) і **Sport Priority / FitnessPriority** (fitnesspriority.com). Обидві системи мають широкий набір функцій для онлайн-запису й обліку відвідувань, однак при цьому не забезпечують у повному обсязі гнучкого керування заявками до тренера, поєднання індивідуальних, групових і самостійних тренувань у єдиному календарі, а також прозорого контролю залишку відвідувань на абонементі. Розглянемо детальніше кожен із цих сервісів, а також їхні сильні та слабкі сторони з погляду функціональності. Нижче наведено таблицю 1.1 із порівнянням двох сервісів-аналогів.

Таблиця 1.1. Порівняльна характеристика сервісів-аналогів

запланованого продукту		
Критерій порівняння	**Mindbody**	**Sport / FitnessPriority**
Адреса вебсервісу	**mindbodyonline.com**	**fitnesspriority.com**
Функціональність	**Розклад занять, онлайн-запис, керування персоналом, оплати**	**CRM для клубу, абонементи, контроль доступу, звіти**
Інтерфейс користувача	**Сучасний, але перевантажений опціями для малих клубів**	**Адміністративно-орієнтований, складніший для відвідувача**
Запис на групове заняття	**Реалізовано**	**Реалізовано**

Запис на індивідуальне тренування з конкретним тренером	**Обмежено, через зовнішні плагіни**	**Обмежено**
Самостійне відвідування за абонементом	**Не виділено як окрема сутність**	**Реалізовано через турнікети, без онлайн-планування**
Заявки до тренера з підтвердженням/відмовою	**Відсутні**	**Відсутні**
Контроль залишку відвідувань абонемента онлайн	**Частково**	**Частково**
Обмеження одного активного абонемента на користувача	**Не передбачено**	**Не передбачено**

Кабінет тренера для керування власним розкладом	**Обмежений**	**Обмежений**
Додаткові функції	**Інтеграція з оплатами, мобільний застосунок**	**Контроль доступу, аналітика клубу**

Джерело: розроблено автором на основі офіційних сайтів сервісів.

На представлених сервісах реалізована базова система онлайн-запису на групові заняття, однак Mindbody орієнтований передусім на великі студії та має складну тарифікацію, а FitnessPriority більшою мірою є адміністративною CRM-системою, у якій відвідувач не отримує повноцінного особистого кабінету з гнучким плануванням. Це створює низку обмежень для невеликих і середніх клубів, яким потрібен баланс між простотою для відвідувача та функціональністю для адміністрації.

Жоден із розглянутих сервісів не виділяє окрему сутність «заявка до тренера», у якій відвідувач пропонує бажані дату й час, а тренер може підтвердити або відхилити запит із відповідним повідомленням клієнту. У запропонованому застосунку ForgeGym така логіка є базовою: вона забезпечує прозорість домовленостей між клієнтом і тренером та автоматично сповіщає відвідувача про результат заявки.

Також жоден із аналогів не поєднує в одному календарі три формати тренувань — індивідуальні з тренером, групові заняття з обмеженою кількістю учасників і самостійні тренування для клієнтів з активним абонементом. Запропонований продукт реалізує єдиний інтерфейс розкладу, у якому відвідувач у

межах одного фільтра обирає категорію (тренажерний зал, стрейчинг, TRX, пілатес, сайклінг) і формат заняття, а система враховує реальну зайнятість тренера в робочих годинах Пн–Пт 09:00–17:00, не дозволяючи створити конфліктний запис.

Окремим недоліком аналогів є відсутність жорсткої бізнес-логіки роботи з абонементом: користувач може мати кілька паралельних карток, відвідування не завжди списуються коректно, а тренер не бачить, чи дійсний абонемент клієнта на момент заняття. У запропонованому продукті реалізовано правило «один активний абонемент на користувача», автоматичне списання відвідування при підтвердженні тренування та неможливість запису без діючого абонемента, що особливо важливо для самостійних тренувань.

Запропонований вебзастосунок суттєво перевищує розглянуті аналоги завдяки поєднанню класичних можливостей (розклад, профілі, оплати) з цільовою бізнес-логікою фітнес-клубу: системою заявок до тренера, єдиним календарем для трьох форматів тренувань, обмеженням групових місць, прозорим обліком абонемента та чіткою рольовою моделлю (адміністратор / тренер / відвідувач). Такий комплекс функцій дозволяє говорити не просто про онлайн-розклад, а про систему, орієнтовану на повний цикл організації тренувань у конкретному клубі.

1.3. Постановка задачі для розробки вебзастосунку фітнес-клубу

Для створення сучасного й функціонального вебзастосунку для управління фітнес-клубом недостатньо реалізувати лише стандартний набір можливостей, таких як сторінка з розкладом і форма запису. Сучасний відвідувач очікує значно більше: простоту у використанні, швидкість, індивідуальний підхід, прозору інформацію про тренера та абонемент, а також зручну взаємодію з адміністрацією клубу. Для досягнення цього на початку розробки важливо чітко визначити, що саме має робити система, які функції вона повинна мати, яким буде інтерфейс і як виглядатиме взаємодія для різних типів користувачів. Запропонований проєкт — це не просто електронний розклад, а інтегрована платформа, яка враховує потреби

кожної ролі та забезпечує максимально зручний спосіб планування, проведення й обліку тренувань.

При створенні такого вебзастосунку з орієнтацією на індивідуальні потреби клієнтів виникає низка складних завдань, кожне з яких має свої переваги та недоліки. Розробка платформи вимагає комплексного підходу, у якому кожна функція повинна бути належним чином інтегрована в систему для забезпечення зручності та коректності бізнес-логіки. Завдання, з якими стикається розробник, є багатокритеріальними та передбачають одночасний облік багатьох умов: зайнятість тренера, місткість групи, статус абонемента, формат заняття та роль користувача. **Метою розробки** є створення вебзастосунку, який дозволяє користувачу швидко та зручно записуватися на тренування з урахуванням індивідуальних потреб і формату заняття, отримувати інформацію про тренерів та їхні спеціалізації, взаємодіяти з тренером через заявки, оформлювати й контролювати абонемент і відстежувати свою історію тренувань. У межах проєкту система повинна виконувати такі **основні функції**:

1. Користувач повинен мати можливість зареєструватися та керувати власним профілем (відвідувач, тренер, адміністратор — з різними рівнями доступу).
2. Усі тренування повинні бути структуровані за категоріями (тренажерний зал, стрейчинг, TRX, пілатес, сайклінг). Кожна категорія повинна мати докладний опис, графік, ціну за групове та індивідуальне заняття, а також ілюстративне зображення.
3. Можливість фільтрувати тренування не лише за категорією, а й за форматом (групове, індивідуальне, самостійне) та за спеціалізаціями тренера.
4. Можливість зберігати історію тренувань відвідувача й переглядати її з фільтрами «Майбутні / Минулі / Групові / Індивідуальні».
5. Можливість перевірити статус заявки до тренера (очікує підтвердження, підтверджена, відхилена) та отримати автоматичне сповіщення про результат.

6. Можливість придбати абонемент із заданою кількістю тренувань на місяць або безлімітний варіант, а також автоматичне списання тренувань після проведення заняття.
7. Перегляд списку власних абонементів і їхніх деталей: дата покупки, тип, залишок відвідувань, термін дії.
8. Сторінка планування тренувань повинна бути інтуїтивно зрозумілою, з адаптивним календарем та зручними фільтрами.
9. На сайті повинен бути окремий кабінет тренера, у якому він самостійно формує розклад групових занять, відмічає свої спеціалізації та обробляє заявки відвідувачів.
10. На сайті повинна бути окрема адміністративна панель, у якій адміністратор керує категоріями тренувань, тренерами, абонементами, сесіями та контентом головної сторінки.

Окрім основного функціоналу, майбутня система повинна відповідати сучасним стандартам зручності, безпеки та ефективності:

- **Зручний інтерфейс.** Сайт повинен бути інтуїтивно зрозумілим, з адаптивною версткою для настільних і мобільних пристроїв.
- **Безпека.** Усі персональні дані користувачів повинні бути захищеними, доступ до функцій — обмеженим відповідно до ролі через JWT-автентифікацію та RBAC.
- **Продуктивність.** Сторінки повинні відкриватися швидко, фільтрація розкладу та зміна параметрів запису — працювати майже миттєво.
- **Цілісність бізнес-логіки.** Система не повинна допускати конфліктних записів (подвійне бронювання тренера, перевищення місткості групи, запис без активного абонемента).

- **Масштабованість.** У майбутньому застосунок має легко розширюватись: додавання нових категорій тренувань, типів абонементів, інтеграцій з онлайн-оплатами та мобільним кабінетом.

Запланований вебзастосунок створюється для широкої аудиторії: від звичайних відвідувачів, які шукають зручний спосіб записатися на тренування, до тренерів, які потребують зрозумілого інструмента ведення власного розкладу, і адміністраторів клубу, яким необхідний єдиний центр керування контентом, абонементами та користувачами. Окрему увагу приділено саме рольовому розмежуванню, оскільки кожна з категорій користувачів має принципово різні сценарії використання.

1.4. Обґрунтування вибору програмного середовища розробки вебзастосунку для фітнес-клубу

Реалізація зазначених бізнес-процесів потребує вирішення низки технічних завдань. Зокрема, важливо розробити модель предметної області (User, TrainingCategory, TrainingSession, TrainingRequest, MembershipPlan, MembershipCard) та узгоджені правила переходів між станами заявок і абонементів. Крім того, необхідно спроєктувати REST API на ASP.NET Core, налаштувати Entity Framework Core та міграції бази даних PostgreSQL, а також реалізувати клієнтський інтерфейс на React (Vite, Tailwind CSS) з адаптивним календарем і фільтрами. Дане технічне завдання охоплює всі ключові вимоги до системи, яка повинна не лише виконувати функції стандартного онлайн-розкладу, а й надавати розширені можливості, орієнтовані на специфіку фітнес-клубу.

Розробка вебзастосунку **ForgeGym** охоплює одночасно серверну частину (ASP.NET Core, C#, Entity Framework Core, PostgreSQL) та клієнтську (React, Vite, JavaScript). Це передбачає роботу з великою кількістю файлів різних типів — класи доменної моделі, контролери API, міграції БД, компоненти інтерфейсу, конфігурація розгортання (render.yaml, Dockerfile). Тому вибір **програмного середовища розробки (IDE)** має забезпечувати підтримку обох мовних стеків, зручну навігацію

по проєкту, інтеграцію з системою контролю версій, терміналом і засобами налагодження.

На сучасному ринку представлені різні середовища для створення вебзастосунків:

- **Microsoft Visual Studio** — потужне інтегроване середовище з розширеними можливостями для .NET (візуальні дизайнери, профілювання, глибока інтеграція з Azure). Проте повна версія має підвищені системні вимоги, а для паралельної роботи з React-фронтендом часто доводиться користуватися додатковими інструментами або окремим вікном редактора.

- **JetBrains Rider** — професійна кросплатформена IDE для .NET із якісним аналізом коду та рефакторингом. Недоліком для навчального проєкту є необхідність комерційної ліцензії (або обмеження пробного періоду), що не завжди зручно для тривалої кваліфікаційної роботи.

- **Visual Studio Code** — легке кросплатформенне редакторське середовище з відкритим кодом, яке за рахунок розширень (extensions) перетворюється на повноцінне середовище для full-stack розробки. Воно однаково добре підходить для редагування .cs, .jsx, .json, SQL-скриптів і markdown-документації диплома в межах одного робочого простору.

- **WebStorm** та інші спеціалізовані IDE для JavaScript орієнтовані переважно на фронтенд і гірше інтегровані з екосистемою .NET без додаткових налаштувань.

Для виконання даної кваліфікаційної роботи основним програмним середовищем обрано **Visual Studio Code** (далі — VS Code). Доцільність цього вибору для проєкту ForgeGym зумовлена такими факторами.

Підтримка backend-частини (.NET). Завдяки розширенню **C# Dev Kit** (або **C#** від Microsoft) VS Code забезпечує підсвічування синтаксису, автодоповнення, навігацію «перейти до визначення», підказки щодо помилок компіляції та інтеграцію з dotnet

CLI. Це дозволяє в одному вікні редагувати контролери (TrainingCategoriesController, TrainingsController), доменні моделі (ForgeGym.Domain) та файли міграцій EF Core.

Підтримка frontend-частини (React). Розширення **ESLint**, підтримка JSX/TS у вбудованому редакторі та інтеграція з **npm** через вбудований термінал дають змогу розробляти сторінки PlanTrainingPage, VisitorDashboard, компонент ProtectedRoute із перевіркою стилю коду та швидким запуском npm run dev (Vite).

Єдиний робочий простір (workspace). У VS Code відкрито кореневу папку репозиторію duplom, що містить рішення .sln, проєкти API, Infrastructure, Web і каталог docs/thesis з діаграмами та текстом роботи. Перемикання між серверним і клієнтським кодом відбувається без зміни програми.

Вбудований термінал і Git. Команды dotnet build, dotnet ef migrations add, dotnet test, git status, git commit виконуються безпосередньо в IDE. Це спрощує цикл «зміна коду → збірка → тест → фіксація в репозиторії», що використовувався під час усієї розробки ForgeGym.

Розширення для документації та діаграм. Підтримка Markdown, PlantUML, попередній перегляд .md-файлів, робота з .puml для UML-діаграм (рис. 2.1.1, 2.2.2, 2.2.3) дозволяють поєднати програмування та оформлення пояснювальної записки в одному середовищі.

Налагодження. Для API налаштовується запуск через launch.json (профіль .NET Core Attach / Launch), для фронтенду — відладка в браузері (розширення для Chrome/Edge) або логування в консолі Vite. Swagger UI у режимі Development доповнює перевірку REST-ендпоінтів без окремого клієнта Postman.

Кросплатформність і доступність. VS Code безкоштовний, працює на Windows (основне середовище розробника проєкту), macOS і Linux, що відповідає вимогам універсальності та економії ресурсів порівняно з повною Visual Studio.

Додаткові інструменти в екосистемі VS Code. У процесі роботи також використовувались:

- **.NET SDK 9** — збірка та запуск API з командного рядка (dotnet run --project ForgeGym.Api);

- **Node.js і npm** — встановлення залежностей і збірка React-застосунку;
- **pgAdmin** або вбудовані засоби перегляду БД — перевірка таблиць PostgreSQL після міграцій;
- **браузер** (Chrome / Edge) — тестування SPA та DevTools для мережевих запитів до API;
- **Git** — облік версій коду (локально та при публікації на GitHub);
- **Docker** (за потреби) — збірка образу API згідно з Dockerfile для Render.

Разом із VS Code ці засоби утворюють **цілісне програмне середовище розробки** вебзастосунку для фітнес-клубу: від написання доменної моделі та REST API до верстки інтерфейсу, тестів (ForgeGym.Tests) і підготовки матеріалів диплома.

Висновок. Вибір **Visual Studio Code** як основного середовища розробки для ForgeGym є обґрунтованим: він поєднує підтримку C#/.NET і React/JavaScript, інтеграцію з dotnet та npm, Git, терміналом і розширеннями для документації, забезпечує продуктивну full-stack роботу над кваліфікаційним проєктом без необхідності використовувати окремі IDE для backend і frontend.

Висновки до розділу 1

У даному розділі було виявлено, що ринок цифрових сервісів для фітнес-індустрії стрімко зростає, а користувачі все більше потребують зручних та індивідуальних інструментів для планування тренувань. Особливий акцент зроблено на важливості онлайн-бронювання, прозорого обліку абонементів і чіткого розмежування ролей у системі, що сьогодні є актуальним і реалізованим лише частково в більшості наявних рішень.

Аналіз наявних платформ, таких як Mindbody та FitnessPriority, показав обмеженість їхньої функціональності в контексті індивідуальної взаємодії клієнта з тренером, відсутність гнучкої системи заявок, єдиного календаря для трьох форматів тренувань (групові, індивідуальні, самостійні) та чіткого правила «один активний

абонемент на користувача». Це відкриває простір для створення сучасного, конкурентного продукту, який враховує специфіку роботи невеликих і середніх фітнес-клубів та потреби різних категорій користувачів.

Також у межах розділу було сформульовано постановку задачі та технічне бачення майбутнього програмного продукту. Визначено функціональні та нефункціональні вимоги до системи, які мають на меті забезпечити інтуїтивну взаємодію, безпеку, швидкість роботи, цілісність бізнес-логіки та масштабованість. Особливу увагу приділено рольовій моделі доступу (адміністратор / тренер / відвідувач) та підтриманню узгодженості даних при бронюванні та обліку абонементів.

Отже, на основі проведеного аналізу сформовано обґрунтовану потребу у створенні вебзастосунку, який стане не просто електронним розкладом, а повноцінним цифровим простором для зручного, безпечного та прозорого управління тренуваннями, тренерами й абонементами в межах сучасного фітнес-клубу.

РОЗДІЛ 2

ПРОЄКТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ФІТНЕС-КЛУБОМ

2.1. Аналіз предметної області програми

Першим кроком у розробці подібної системи є аналіз предметної області, тобто вивчення того, як повинна працювати система, які її основні складові, які дані вона обробляє, які функції виконує і які обмеження має. Це дозволяє побудувати логічну та зрозумілу структуру, яка в майбутньому стане основою для програмної реалізації. У процесі аналізу було визначено ключові об’єкти, з якими взаємодіють користувачі та адміністрація клубу. Серед них:

- користувач-системи(відвідувач,тренер,адміністратор);
- профіль тренера та профіль відвідувача;
- категорія тренування (вид заняття);
- сесія тренування (розклад: групова, індивідуальна, самостійна);
- бронювання відвідувача на сесію;
- заявка відвідувача до тренера на індивідуальне заняття;
- тарифний план абонемента та видана картка абонемента;
- сповіщення користувача (про результат заявки тощо).

Процес взаємодії з вебзастосунком фітнес-клубу є послідовним і структурованим. Його метою є забезпечити зручний запис на тренування, прозорий облік абонементів і узгоджену взаємодію між відвідувачем, тренером та адміністрацією. У межах функціональної моделі основну роль відіграє автоматизація обліку: перевірка наявності активного абонемента, обмеження кількості учасників у груповому занятті, перевірка зайнятості тренера в робочих годинах, списання відвідування з абонемента після підтвердження тренування або заявки. Сучасний клієнт очікує не лише статичного розкладу, а й можливості

швидко обрати формат заняття (групове, індивідуальне, самостійне), фільтрувати тренерів за спеціалізаціями та отримувати сповіщення про зміну статусу заявки.

Для наочності нижче наведено UML-діаграму варіантів використання, яка відображає основні взаємодії ролей із системою (рис. 2.1).

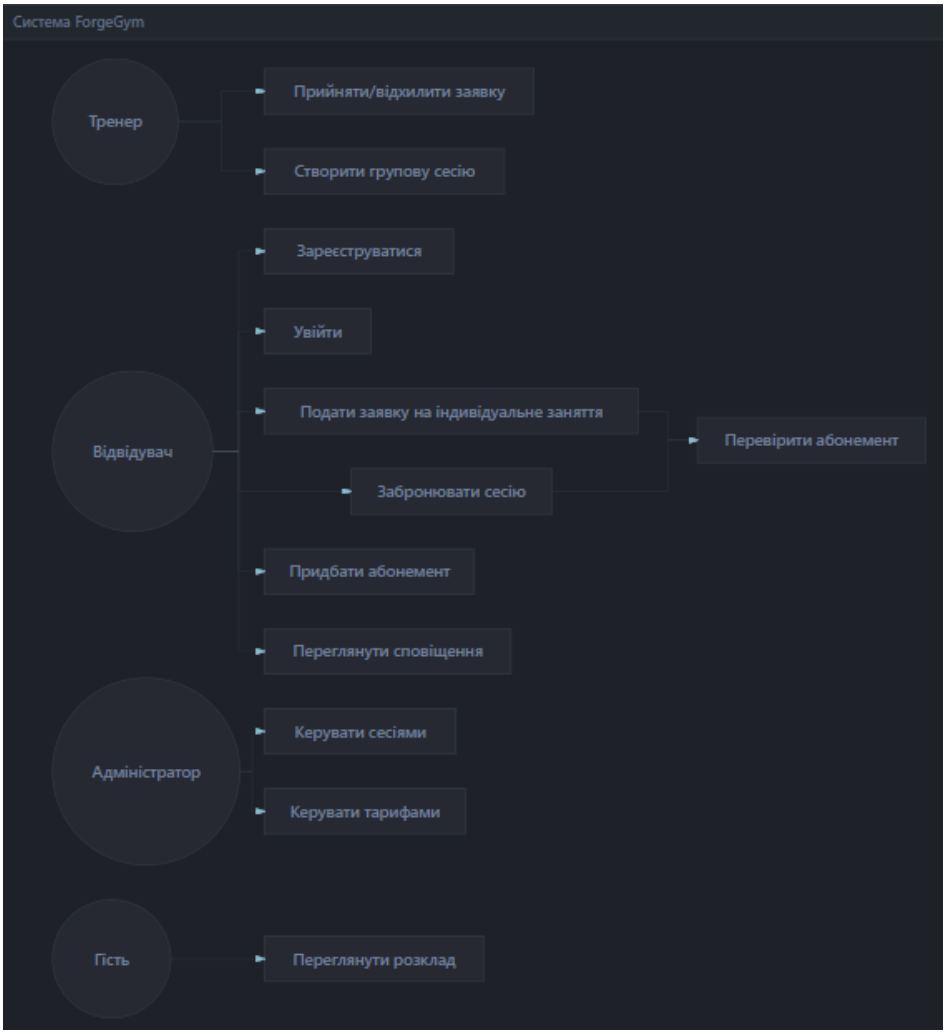

Рис. 2.1 Варіанти використання вебзастосунку ForgeGym

Джерело: створено автором

Інформаційна модель демонструє, як користувач взаємодіє із системою залежно від ролі. UML-діаграма діяльності (рис. 2.2) ілюструє типовий сценарій відвідувача: від перегляду каталогу категорій тренувань до запису на заняття або подання заявки тренеру.

Рис. 2.2. UML-діаграма діяльності відвідувача
у вебзастосунку ForgeGym

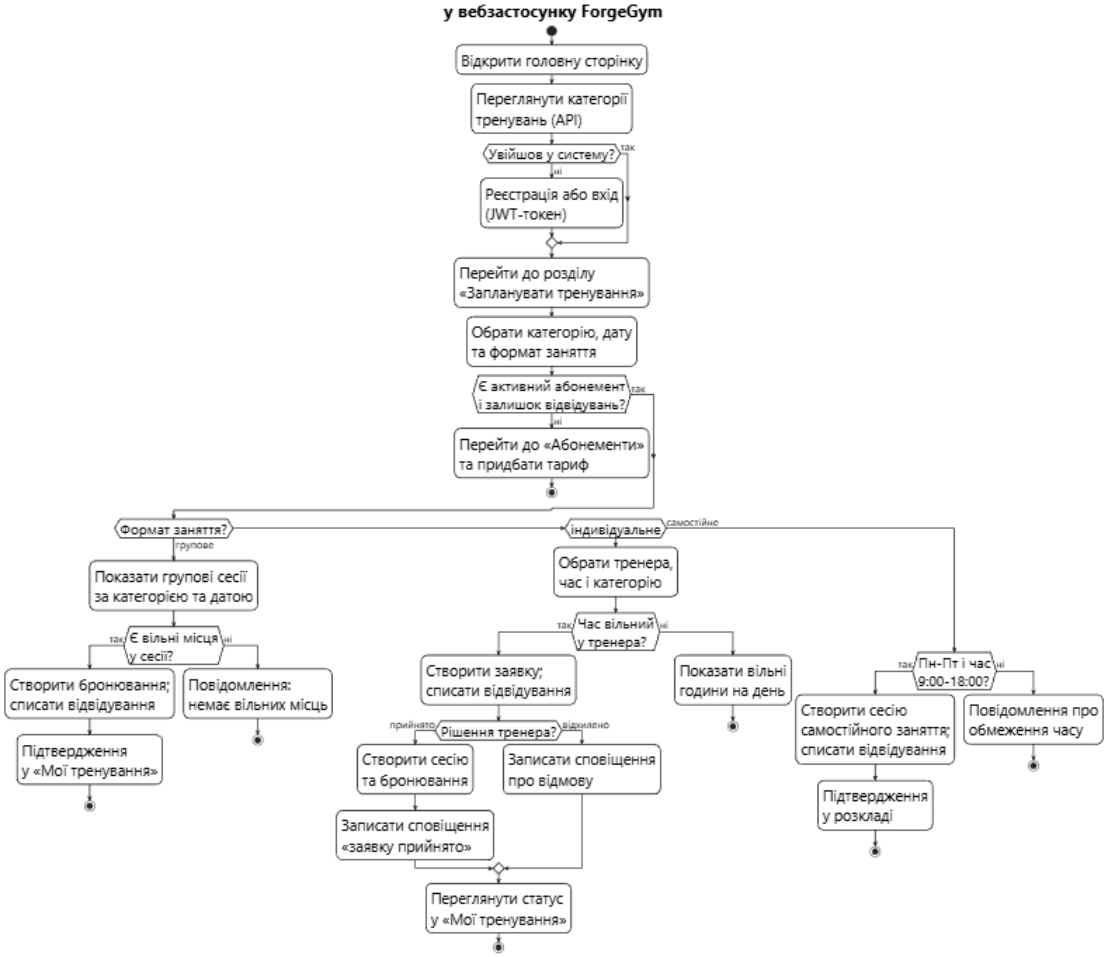

Рис. 2.2 UML-діаграма діяльності відвідувача у вебзастосунку ForgeGym

Джерело: створено автором

Типовий процес для відвідувача можна описати такими етапами:

1. **Відкриття публічної частини сайту.** Користувач переглядає головну сторінку з описом категорій тренувань, цін і графіку (дані завантажуються з API).
2. **Реєстрація або вхід.** Створення облікового запису або автентифікація за допомогою JWT; залежно від ролі відкривається відповідний кабінет (відвідувач, тренер, адміністратор).

3. **Вибір сценарію.** Відвідувач переходить до розділів «Тренери», «Запланувати тренування», «Мої тренування», «Абонементи» тощо.
4. **Фільтрація та вибір параметрів.** Під час планування тренування застосовуються фільтри за категорією, форматом заняття (групове / індивідуальне / самостійне) та спеціалізаціями тренера; для індивідуальних заявок перевіряється доступність обраного часу.
5. **Бронювання або заявка.** Для групової сесії створюється запис бронювання з урахуванням місткості; для індивідуального формату — заявка до тренера з очікуванням підтвердження або відмови; для самостійного заняття — створення сесії за наявності активного абонемента.
6. **Облік абонемента.** Система перевіряє наявність одного активного абонемента, залишок відвідувань (для планів з лімітом) і списує відвідування відповідно до бізнес-правил.
7. **Сповіщення.** Після зміни статусу заявки користувач отримує запис у таблиці сповіщень і може переглянути його в інтерфейсі.
8. **Дії тренера та адміністратора.** Тренер керує власним розкладом групових занять, спеціалізаціями та обробляє заявки; адміністратор веде категорії, користувачів, плани абонементів і сесії через адмін-панель.

2.2. Створення та моделювання бази даних застосунку

База даних — це організоване електронне середовище для збереження та обробки інформації. У контексті вебзастосунку для управління фітнес-клубом вона зберігає дані про користувачів, розклад тренувань, бронювання, заявки, абонементи та сповіщення. Основна мета — забезпечити швидкий доступ, цілісність даних і підтримку транзакцій при одночасних записах кількох відвідувачів на одну групову сесію.

Для реалізації обрано СУБД **PostgreSQL** як надійну реляційну систему з підтримкою складних запитів і зовнішніх ключів. Зв’язок між застосунком і базою реалізовано через **Entity Framework Core** — ORM, що дозволяє описувати сутності мовою C# і застосовувати **міграції** для еволюції схеми без втрати історії змін.

Під час проєктування виділено такі основні сутності (відповідають доменним моделям у проєкті ForgeGym.Domain):

1. **User** — обліковий запис користувача: електронна пошта (Email), телефон (Phone), ім’я та прізвище (FirstName, LastName), роль (Role: відвідувач, тренер, адміністратор), хеш пароля (PasswordHash), набір прав адміністратора (AdminPermissions). Зв’язані профілі: TrainerProfile та VisitorProfile.
2. **TrainerProfile** (профіль тренера): ідентифікатор користувача (UserId), біографія (Bio), стаж у роках (ExperienceYears), сертифікати (Certifications), рядок спеціалізацій (Specializations), URL фото (PhotoUrl).
3. **VisitorProfile** — мінімальний профіль відвідувача, прив’язаний до UserId.
4. **TrainingCategory** — категорія тренування: назва (Name), ключ іконки (IconKey), опис (Description), URL зображення (ImageUrl), текст розкладу (Schedule), ціни групового та індивідуального заняття в гривнях (PriceGroupUah, PriceIndividualUah). Містить колекцію сесій.
5. **TrainingSession** — конкретне заняття в розкладі: посилання на категорію (CategoryId), опційно тренер (TrainerId), ознака самостійного заняття (IsSelfWorkout), час початку (StartsAt), тривалість у хвилинах (DurationMinutes), ознака групового заняття (IsGroup), місткість (Capacity), ціна (PriceUah). Містить колекцію бронювань.
6. **Booking** — бронювання відвідувача на сесію: SessionId, VisitorId, статус (Status), час скасування (CancelledAt), посилання на картку абонемента, з якої списано відвідування (UsedMembershipCardId).
7. **TrainingRequest** — заявка відвідувача на індивідуальне тренування: VisitorId, CategoryId, бажаний час (DesiredAt), тривалість і ціна, бажаний тренер (PreferredTrainerId), статус (Status), прийнятий тренер і час прийняття (AcceptedTrainerId, AcceptedAt), створена після підтвердження сесія (CreatedSessionId), нотатки (Notes), посилання на абонемент (UsedMembershipCardId). Містить колекцію відмов тренерів (Declines).
8. **TrainingRequestDecline** — відмова тренера на заявку: RequestId, TrainerId, причина (Reason).

9. **MembershipPlan** — тариф абонемента: код і назва (Code, Name), ціна (PriceUah), тривалість у днях (DurationDays), ліміт відвідувань (VisitsAllowed), дозволений діапазон годин (AllowedHourFrom, AllowedHourTo), ознаки «з тренером» та «активний план» (IncludesTrainer, IsActive).
10. **MembershipCard** — виданий абонемент відвідувачу: VisitorId, PlanId, період дії (ActiveFrom, ActiveTo), залишок відвідувань (RemainingVisits), сплачена сума (PriceUahPaid).
11. **Notification** — сповіщення користувачу: UserId, тип (Kind), заголовок і текст (Title, Body), пов’язаний URL (RelatedUrl), час прочитання (ReadAt).

Усі сутності успадковують базовий клас **EntityBase** з полями Id (Guid), CreatedAt, UpdatedAt.

Зв’язки між таблицями реалізовано через зовнішні ключі та навігаційні властивості EF Core: **один-до-багатьох** (категорія — сесії, користувач — бронювання, план — картки абонементів тощо), **багато-до-одного** (бронювання — сесія та відвідувач). Для заявок передбачено окрему таблицю відмов тренерів, що дозволяє зберігати історію відхилень.

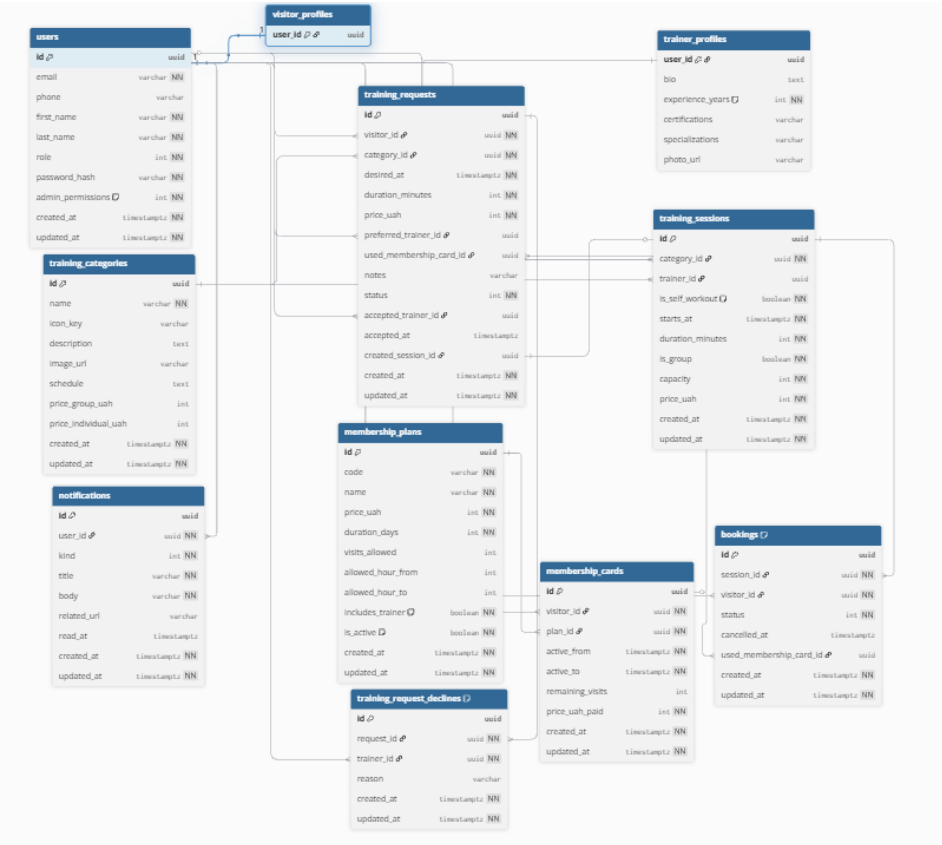

Рис. 2.3 ER-діаграма основних таблиць та зв’язків бази даних ForgeGym

Джерело: розроблено автором

Приклад опису сутності на рівні коду наведено для **TrainingCategory** — однієї з центральних моделей каталогу тренувань.

Лістинг 2.1. Модель сутності TrainingCategory (фрагмент проєкту ForgeGym.Domain).

```
public sealed class TrainingCategory : EntityBase
{
    public required string Name { get; set; }
    public string? IconKey { get; set; }
    public string? Description { get; set; }
    public string? ImageUrl { get; set; }
    public string? Schedule { get; set; }
    public int? PriceGroupUah { get; set; }
    public int? PriceIndividualUah { get; set; }
    public List<TrainingSession> Sessions { get; set; } = [];
}
```

2.3. Проєктування архітектури серверної частини та REST API

Після аналізу предметної області та проєктування бази даних наступним етапом є визначення архітектури серверної частини і інтерфейсу взаємодії між клієнтським застосунком (React) та сервером. Мета цього підрозділу — описати, як логіка фітнес-клубу розподіляється між програмними шарами, які ресурси надає REST API і як забезпечується безпечний доступ за ролями. Отримана модель є безпосередньою основою для програмної реалізації, розглянутої в розділі 3.

Багатошарова архітектура серверної частини

Серверна частина ForgeGym побудована за принципом розділення відповідальності (Separation of Concerns): кожен шар виконує вузько визначені завдання і залежить лише від нижчих рівнів абстракції. Це спрощує супровід, тестування та подальше розширення функціоналу (наприклад, додавання нових

типів занять або звітів для адміністратора).

ForgeGym.Domain — доменний шар. Містить сутності предметної області (User, TrainingCategory, TrainingSession, Booking, TrainingRequest, MembershipPlan, MembershipCard, Notification тощо), переліки (UserRole, BookingStatus, TrainingRequestStatus, NotificationKind) та допоміжні правила валідації вхідних даних (InputValidators). Шар не залежить від бази даних, веб-фреймворку чи UI — це «ядро» бізнес-змісту системи, узгоджене з ER-моделлю з п. 2.2.

ForgeGym.Application — шар застосунку (абстракції). Визначає контракти, через які верхні рівні звертаються до інфраструктури: інтерфейс IAppDbContext (набір DbSet і SaveChangesAsync) та IJwtTokenService для формування токена автентифікації.

Такий підхід дозволяє контролерам API не прив’язуватися безпосередньо до конкретної реалізації DbContext і спрощує модульне тестування.
ForgeGym.Infrastructure — інфраструктурний шар. Реалізує AppDbContext (Entity Framework Core + PostgreSQL), конфігурацію зв’язків між таблицями (OnModelCreating), міграції схеми БД, сервіс JwtTokenService, реєстрацію залежностей у DependencyInjection. Усі зовнішні технічні деталі (Npgsql, JWT, хешування паролів через BCrypt на рівні API) зосереджені тут або в шарі API.

ForgeGym.Api — шар представлення (Presentation / Web API). Містить ASP.NET Core контролери, DTO запитів і відповідей, допоміжні класи (CurrentUser, MembershipHelper), конфігурацію Program.cs (CORS, JWT, Swagger, підключення до БД), початкове наповнення БД (DatabaseSeeder). Саме цей шар приймає HTTP-запити від клієнта і повертає відповіді у форматі JSON.

Клієнтська частина (ForgeGym.Web, React + Vite) взаємодіє з ForgeGym.Api лише через HTTP; прямого доступу до бази даних з браузера немає. Це відповідає класичній схемі клієнт — сервер і вимогам безпеки.

Принципи побудови REST API

Інтерфейс серверної частини спроєктовано як REST API (Representational State Transfer): ресурси (категорії, сесії, абонементи, заявки) доступні за унікальними URL, операції над ними виконуються стандартними HTTP-методами.

Основні принципи, adopted у ForgeGym:

1. Ресурсна адресація. URL відображають сутності предметної області, наприклад: /api/trainings/categories, /api/trainings/sessions, /api/memberships/plans, /api/trainings/requests.
2. Використання HTTP-методів за призначенням. GET — читання, POST — створення або дія (бронювання, прийняття заявки), PUT/PATCH — оновлення, DELETE — видалення.
3. Формат обміну JSON. Тіло запитів і відповідей — application/json; клієнт передає дані в об’єктах, сервер повертає проєкції сутностей або анонімні DTO без зайвого навантаження клієнта повними графами EF.
4. Статусні коди HTTP. 200 OK — успіх, 201 Created — створення ресурсу, 400 Bad Request — помилка валідації, 401 Unauthorized / 403 Forbid — проблеми автентифікації чи прав, 404 Not Found — відсутній запис, 409 Conflict — порушення бізнес-правила (немає місць на сесію, активний абонемент уже існує, тренер зайнятий).
5. Безстанність (stateless). Сервер не зберігає сесію користувача в пам’яті; після входу клієнт надсилає JWT у заголовку Authorization: Bearer <token> на кожен захищений запит.
6. Публічні та захищені ендпоінти. Читання каталогу категорій, тарифів, розкладу сесій і списку тренерів доступне без токена ([AllowAnonymous]); операції кабінету відвідувача, тренера та адміністратора вимагають автентифікації та відповідної ролі.

Документування API в режимі розробки передбачено через Swagger UI (/swagger), що прискорює перевірку ендпоінтів до інтеграції з React-фронтендом.

Структура контролерів і груп ресурсів

Логіка REST API розбита на тематичні контролери, кожен з яких відповідає за фрагмент функціоналу з діаграми варіантів використання (рис. 2.1.1).

Контролери не використовують окремий шар репозиторіїв: доступ до даних здійснюється через IAppDbContext, інжектований у конструктор. Це прийнятне рішення для проєкту середньої складності — воно зменшує кількість класів-

обгорток і зберігає прозорість запитів LINQ у місці обробки HTTP-запиту.

Лістинг 2.2. Реєстрація сервісів інфраструктури та підключення БД (фрагмент DependencyInjection.cs).

```
services.AddDbContext<AppDbContext>(options =>
{
   options.UseNpgsql(connectionString);
});
services.AddScoped<IAppDbContext>(sp => sp.GetRequiredService<AppDbContext>());
services.AddSingleton<IJwtTokenService, JwtTokenService>();
```

. Проєктування основних ендпоінтів API

Проєктування автентифікації та авторизації

Безпека API спроєктована на основі JWT Bearer і ролей ASP.NET Core. Сценарій входу:

7. Клієнт надсилає POST /api/auth/login з email і паролем.
8. Сервер перевіряє хеш пароля (BCrypt), формує JWT із claims: sub (Id користувача), email, role, імена.
9. Клієнт зберігає токен і додає заголовок Authorization до наступних запитів.
10. Middleware UseAuthentication / UseAuthorization перевіряє підпис і термін дії токена; атрибут [Authorize(Roles = "...")] обмежує доступ до методів контролера.

Розмежування відповідає ролям з п. 2.1: Visitor — бронювання та абонементи; Trainer — розклад і заявки; Admin — довідники, тарифи, користувачі. Додатково для адміністратора передбачено бітову маску AdminPermissions (наприклад, право призначати тренерів) з перевіркою в AdminUsersController.

Взаємодія з клієнтською частиною та обробка помилок

SPA на React звертається до API через модуль api/client.js: базова адреса задається змінною VITE_API_URL, токен читається з localStorage. При помилках сервер повертає JSON із полем message (українськомовні тексти для бізнес-порушень), клієнт відображає їх користувачу (наприклад, відсутність місць на занятті або

конфлікт часу тренера).

CORS налаштовано для дозволених origin фронтенду (локально в Development — вільніше, у production — список з конфігурації Render/Vercel). Окремі службові маршрути: GET /health для перевірки доступності API при хмарному розгортанні.
Узгодження з моделлю даних і функціональними вимогами
Спроєктована архітектура та REST API забезпечують:

- повне покриття сутностей з п. 2.2 відповідними групами ендпоінтів;
- реалізацію бізнес-правил (абонемент, місткість, робочі години) на рівні методів контролерів перед SaveChangesAsync;
- чітке розділення публічного каталогу та захищених кабінетних операцій;
- можливість незалежної розробки та розгортання фронтенду (Vercel) і backend (Render) при збереженні єдиного контракту JSON API.

Таким чином, підрозділ 2.3 завершує проєктування серверної частини: після предметної моделі і структури даних визначено програмну архітектуру шарів і зовнішній інтерфейс REST, що безпосередньо використовується при реалізації ForgeGym у розділі 3

Висновки до розділу 2

У другому розділі кваліфікаційної роботи виконано комплексне проєктування серверної частини вебзастосунку для управління фітнес-клубом ForgeGym. Розглянуто предметну область як сукупність бізнес-процесів клубу: облік відвідувачів і персоналу, формування розкладу занять різних форматів, бронювання місць, подання та обробка заявок на індивідуальні тренування, продаж і контроль абонементів, інформування користувачів про зміну статусу операцій. Такий підхід дозволив відокремити змістовні об’єкти предметної області від технічних деталей реалізації та сформувати логічну основу для подальшої розробки REST API і клієнтського застосунку.

У підрозділі 2.1 здійснено аналіз предметної області та описано ролі учасників системи — відвідувача, тренера та адміністратора. Визначено, що кожна роль має

власний набір цілей і обмежень: відвідувач потребує зручного запису на заняття та прозорого обліку абонемента; тренер — інструментів для керування розкладом і відповіді на заявки; адміністратор — централізованого керування довідниками (категорії, тарифи, користувачі, сесії). Окремо зафіксовано бізнес-правила, які мають виконуватися автоматично: наявність активного абонемента та залишку відвідувань, контроль місткості групових занять, перевірка зайнятості тренера в робочих годинах, списання відвідування при підтвердженні запису або поданні заявки. Ці правила визначають не лише логіку серверної частини, а й вимоги до перевірок на рівні API та інтерфейсу.

Для наочної фіксації функціональних вимог побудовано UML-діаграму варіантів використання , яка узагальнює основні сценарії взаємодії ролей із системою: реєстрація та автентифікація, перегляд каталогу тренувань і тренерів, придбання абонемента, бронювання групової сесії, подання заявки на індивідуальне заняття, самостійне тренування, адміністрування довідників і розкладу. Діаграма дала можливість перевірити повноту функціоналу ще до етапу програмування та уникнути пропуску критичних операцій (наприклад, обробки заявок тренером або роботи зі сповіщеннями).

Інформаційна модель поведінки відвідувача представлена UML-діаграмою діяльності і розгорнута у вигляді восьми послідовних етапів — від перегляду публічної частини сайту до отримання сповіщення про результат заявки. Таке моделювання важливе для узгодження серверної логіки з клієнтськими сценаріями: кожен етап діаграми відповідає конкретним запитам до API (завантаження категорій, автентифікація JWT, бронювання, створення заявки, перевірка абонемента) та окремим розділам інтерфейсу («Запланувати тренування», «Мої тренування», «Абонементи»). Окремо враховано паралельні процеси з боку тренера та адміністратора, що підкреслює багаторольовий характер системи.

У підрозділі 2.2 виконано проєктування бази даних. Обґрунтовано вибір PostgreSQL як надійної реляційної СУБД із підтримкою зовнішніх ключів, транзакцій і складних запитів, що необхідно при одночасному записі кількох відвідувачів на одну групову сесію. Зв’язок між застосунком і сховищем даних

передбачено через Entity Framework Core у підході Code First: доменні моделі описуються класами C#, а еволюція схеми забезпечується міграціями без втрати історії змін структури БД.

Детально описано одинадцять основних сутностей, які відображають ключові об’єкти предметної області: користувач і профілі, категорія та сесія тренування, бронювання, заявка та відмова тренера, тарифний план і картка абонемента, сповіщення. Для більшості сутностей застосовано спільний базовий клас EntityBase з полями Id, CreatedAt, UpdatedAt, що спрощує облік часу створення та оновлення записів; профілі тренера й відвідувача мають первинний ключ UserId, що відповідає зв’язку один-до-одного з обліковим записом. Наведено ER-діаграму (рис. 2.3), яка візуалізує таблиці та зв’язки між ними (один-до-багатьох, багато-до-одного), включно з окремою таблицею відмов тренерів на заявки для збереження історії відхилень. Приклад доменної моделі TrainingCategory (лістинг ілюструє перехід від проєктної сутності до програмного коду в збірці ForgeGym.Domain.

У результаті проєктування сформовано цілісну модель даних і функціональних вимог, яка:

- забезпечує збереження всіх сутностей, зазначених у аналізі предметної області;
- підтримує три формати занять (групове, індивідуальне, самостійне) та різні сценарії запису;
- дозволяє реалізувати облік абонементів і автоматичні перевірки перед виконанням операцій;
- передбачає механізм сповіщень користувачів про зміну статусу заявок;
- розмежовує права доступу за ролями на рівні даних і майбутніх API-ендпоінтів.

Отримані результати розділу 2 є методологічною та технічною основою для наступних етапів роботи: реалізації REST API на ASP.NET Core, побудови клієнтської частини на React, налаштування автентифікації JWT, розгортання системи в хмарному середовищі та тестування сценаріїв, описаних на діаграмах. Без

завершеного проєктування серверної частини неможливо коректно реалізувати CRUD-операції, бізнес-правила запису на тренування та інтеграцію з інтерфейсом користувача; тому висновки другого розділу підтверджують доцільність обраної архітектури даних і повноту підготовки до практичної реалізації вебзастосунку ForgeGym.

РОЗДІЛ 3

ПРИКЛАДНА ЧАСТИНА РОЗРОБКИ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ФІТНЕС-КЛУБОМ

3.1. Технічний стек та технології розробки

Основною мовою програмування обрано C# у поєднанні з ASP.NET Core 9 — платформою для створення кросплатформених REST API. Такий вибір дозволяє реалізувати серверну логіку фітнес-клубу з високою продуктивністю, вбудованою підтримкою автентифікації, валідації та розгортання в хмарі.

Архітектура серверної частини побудована за принципом розділення на шари:

- ForgeGym.Domain — доменні сутності (User, TrainingSession, MembershipCard тощо), переліки ролей і статусів;
- ForgeGym.Application — абстракції (IAppDbContext, IJwtTokenService);
- ForgeGym.Infrastructure — реалізація доступу до БД (EF Core, PostgreSQL), JWT, міграції;
- ForgeGym.Api — HTTP-контролери, конфігурація безпеки, Swagger.

Клієнтська частина реалізована як односторінковий застосунок (SPA) на React 19 зі збірником Vite і стилізацією Tailwind CSS. Взаємодія з API відбувається через fetch із передачею JWT у заголовку Authorization: Bearer.

Для зберігання даних використано PostgreSQL — реляційну СУБД із підтримкою зовнішніх ключів, транзакцій і складних запитів. Доступ до бази забезпечує Entity Framework Core 9 у підході Code First: моделі описуються класами C#, схема оновлюється міграціями (dotnet ef migrations add, dotnet ef database update). Адміністрування БД здійснюється в pgAdmin або через консоль Render.

Автентифікація реалізована на основі JWT (JSON Web Token): після входу клієнт зберігає токен у localStorage і додає його до кожного захищеного запиту. Паролі хешуються за допомогою BCrypt.

Розгортання виконано у хмарі: API — Render (Docker, PostgreSQL Frankfurt), фронтенд — Vercel; конфігурація описана у render.yaml. Для локальної та хмарної розробки використовується Visual Studio / Visual Studio Code (або Cursor) із підтримкою .NET SDK 9.

Компонент	Технологія
Мова сервера	C# 12 / .NET 9
API	ASP.NET Core Web API
ORM	Entity Framework Core 9
СУБД	PostgreSQL (Npgsql)
Клієнт	React 19, Vite 8, React Router 7
Автентифікація	JWT Bearer, BCrypt
Документація API	Swagger (режим Development)
Хостинг	Render (API + БД), Vercel (Web)

Табл. 3.1. Технологічний стек ForgeGym

3.2. Реалізація CRUD-операцій для управління даними

У межах ForgeGym CRUD-операції (Create, Read, Update, Delete) реалізовані для ключових сутностей: категорії тренувань, тарифні плани абонементів, сесії розкладу, користувачі (адмін-панель). На відміну від класичного шаблону «репозиторій», доступ до даних здійснюється через інтерфейс IAppDbContext, який інжектується в контролери — це спрощує код і зберігає можливість тестування абстракції контексту.

Створення (Create)

Додавання нового запису виконується методами Add контексту EF Core та

SaveChangesAsync. Приклад — створення категорії тренування адміністратором (листинг 3.1).

Лістинг 3.1. Створення категорії тренування (TrainingCategoriesController, метод Create).

```
var category = new TrainingCategory
{
    Name = name,
    IconKey = string.IsNullOrWhiteSpace(request.IconKey) ? null :
request.IconKey.Trim(),
    Description = NullIfBlank(request.Description),
    ImageUrl = NullIfBlank(request.ImageUrl),
    Schedule = NullIfBlank(request.Schedule),
    PriceGroupUah = request.PriceGroupUah,
    PriceIndividualUah = request.PriceIndividualUah
};

_db.TrainingCategories.Add(category);
await _db.SaveChangesAsync(ct);

return CreatedAtAction(nameof(GetById), new { id = category.Id }, new { ... });
```

Перед збереженням виконується перевірка унікальності назви та обов’язковості полів. Аналогічно реалізовано POST для тарифних планів (MembershipPlansController) та сесій розкладу (TrainingsController).

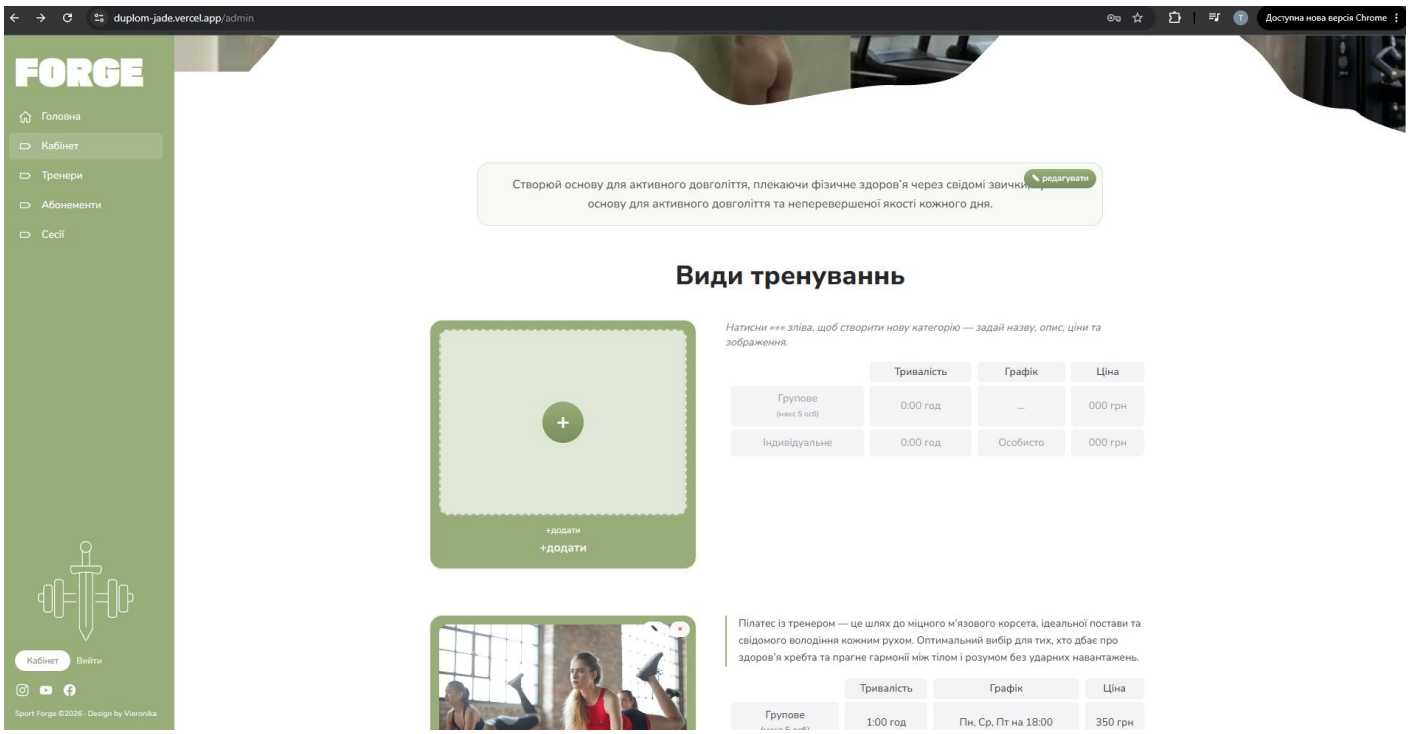

Рис. 3.1 Вигляд сторінки додавання/редагування головної сторінки в адмін-панелі

Джерело: створено автором

Читання (Read)

Операції читання — публічні GET-запити без авторизації (категорії, розклад сесій, список тренерів) та захищені запити для особистого кабінету (мої бронювання, абонементи, сповіщення). Для оптимізації застосовується AsNoTracking() і проєкції Select, що повертають лише потрібні поля (листинг 3.2).

Лістинг 3.2. Отримання списку категорій тренувань.

```
var categories = await _db.TrainingCategories
    .AsNoTracking()
    .OrderBy(x => x.Name)
    .Select(x => new
    {
        x.Id, x.Name, x.IconKey, x.Description,
        x.ImageUrl, x.Schedule, x.PriceGroupUah, x.PriceIndividualUah
    })
    .ToListAsync(ct);
return Ok(categories);
```

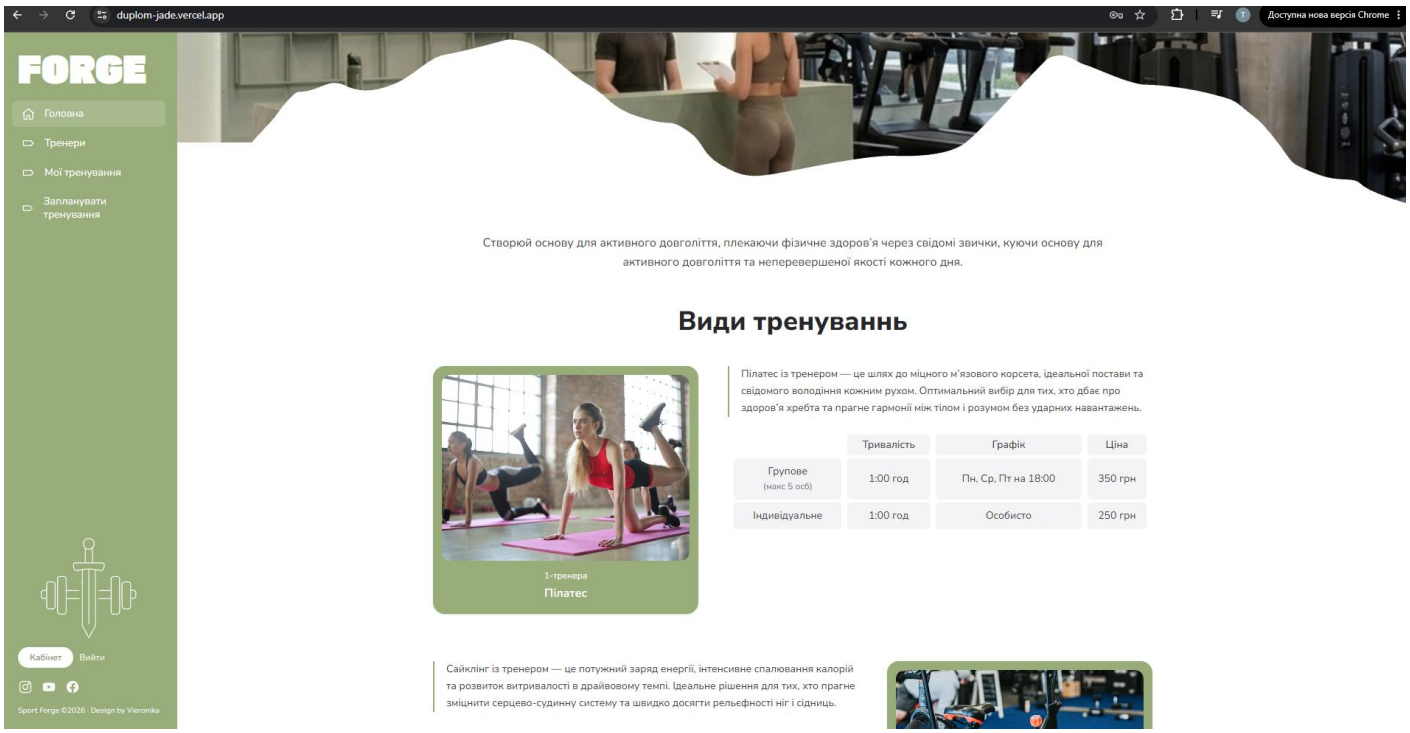

Рис. 3.2 Вигляд головної сторінки з категоріями тренувань

Джерело: створено автором

Оновлення (Update)

Методи PUT завантажують сутність за Id, оновлюють поля з тіла запиту, встановлюють UpdatedAt і викликають SaveChangesAsync. Доступ обмежено роллю Admin (категорії, плани) або Admin/Trainer (сесії, профіль тренера).

Видалення (Delete)

Видалення сесій і категорій доступне адміністратору; перед видаленням перевіряється наявність запису. Для зв'язаних даних у AppDbContext налаштовані правила OnDelete: Cascade, Restrict або SetNull залежно від бізнес-логіки (наприклад, видалення категорії з активними сесіями блокується через Restrict).

Усі CRUD-операції інтегровані в адмін-панель React (/admin, /admin/plans, /admin/sessions) та публічні розділи для відвідувачів.

3.3. Планування тренувань із фільтрацією за форматом і категорією

Ключовою функцією для відвідувача є сторінка «Запланувати тренування» (PlanTrainingPage.jsx). Користувач обирає:

- категорію заняття (завантажується з GET /api/trainings/categories);
- формат: групове, індивідуальне або самостійне;
- дату та час у межах робочого графіка клубу (9:00–18:00, для самостійних — лише Пн–Пт).
- На клієнті застосовується фільтрація:
- групові сесії — з масиву training_sessions за обраною категорією та датою, з перевіркою вільних місць (capacity - bookedCount);
- індивідуальні — список тренерів, у яких у Specializations є код, що відповідає категорії;
- самостійні — перевірка наявності активного абонемента перед відправкою запиту.

Параметри передаються на сервер відповідним POST-запитом: book, training-requests або self-bookings (див. розд. 3.5).

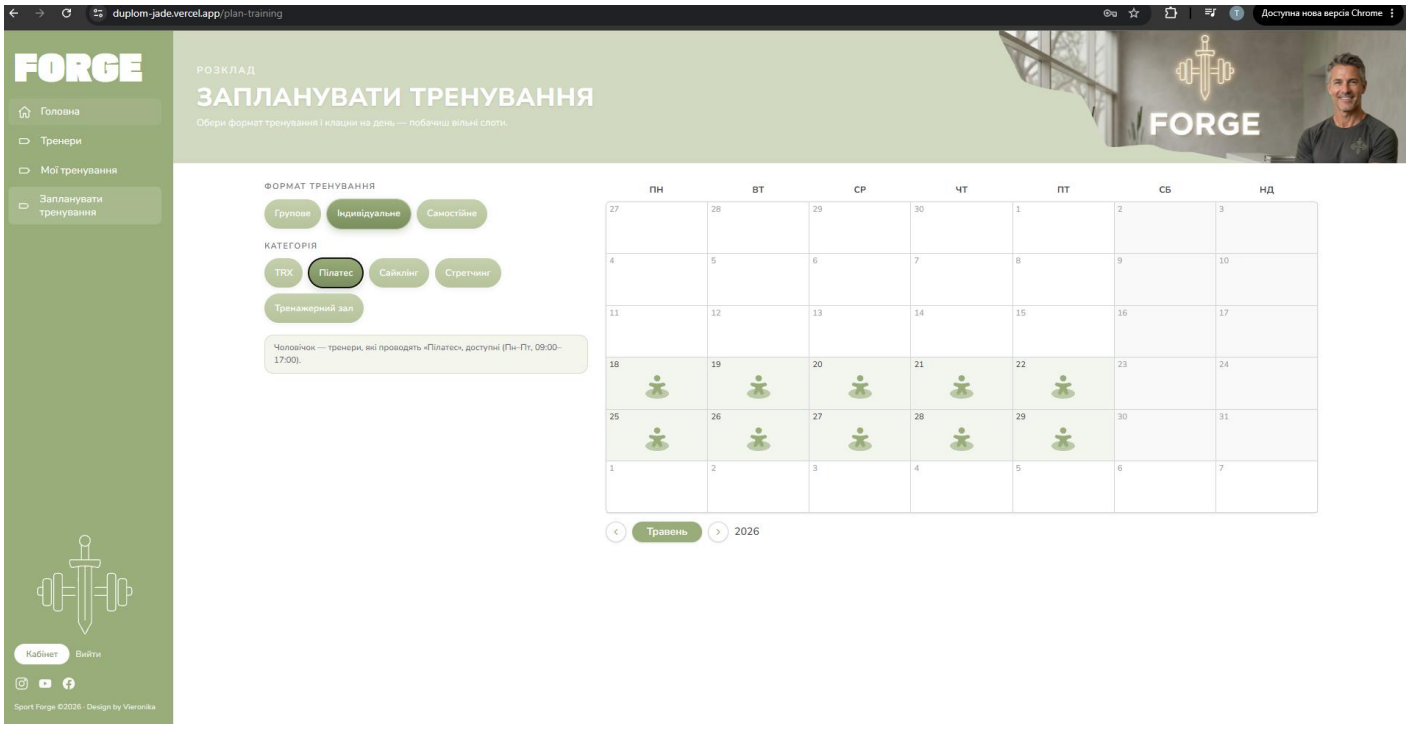

Рис. 3.3 Вигляд сторінки планування тренування з фільтрами

Джерело: створено автором

Фільтрація тренерів за спеціалізаціями та перевірка доступності часу

Для індивідуальних занять реалізовано механізм підбору тренера за спеціалізаціями (рядок Specializations у TrainerProfile, коди на кшталт gym, pilates, stretching). На

клієнті функції categoryToSpecCode та trainerHasSpec відсіюють тренерів, які не ведуть обрану категорію.

Перевірка зайнятості часу виконується на сервері в TrainingRequestsController: при створенні заявки аналізуються накладення з існуючими сесіями та заявками тренера; у разі конфлікту повертається HTTP 409 із списком вільних годин (availableHours).

Лістинг 3.3. Запит доступності тренера на дату.

GET /api/trainings/requests/trainers/{trainerId}/availability?date=2026-05-20&durationMinutes=60

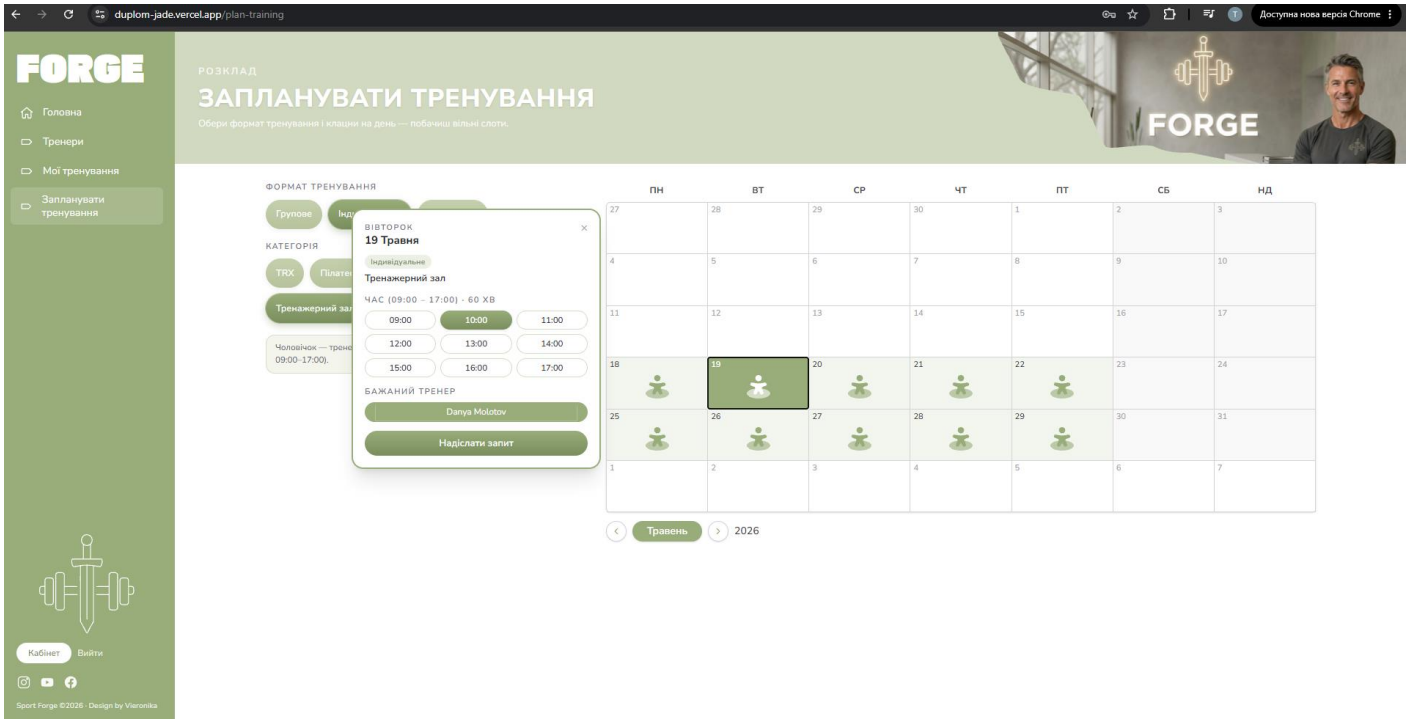

Рис. 3.4 Вибір тренера та часового слоту на сторінці планування

Джерело: створено автором

Публічна сторінка «Тренери» (PublicTrainersPage) відображає профілі з фото, стажем і спеціалізаціями без необхідності входу в систему.

Облік абонементів і бізнес-правила при записі

Система перевіряє наявність одного активного абонемента та залишок відвідувань перед бронюванням, поданням заявки або самостійним заняттям. Логіка зосереджена в MembershipHelper та методах TrainingsController / TrainingRequestsController.

Основні правила:

1. Без активної картки MembershipCard запис неможливий (відповідь 409 із повідомленням для користувача).
2. Якщо план має ліміт VisitsAllowed, при успішній операції поле RemainingVisits зменшується на 1.
3. При скасуванні бронювання відвідування повертається на картку (якщо було списано).
4. Для групових сесій перевіряється bookedCount < capacity; дубльоване бронювання на ту саму сесію заборонене унікальним індексом (SessionId, VisitorId).
5. Самостійне заняття дозволене лише в робочі дні та години, без перетину з іншими бронюваннями відвідувача.

Лістинг 3.4. Придбання абонемента відвідувачем.

```
[HttpPost("purchase")]
[Authorize(Roles = "Visitor")]
public async Task<IActionResult> Purchase(PurchaseMembershipRequest request, ...)
{
   var plan = await _db.MembershipPlans.FirstOrDefaultAsync(x => x.Id ==
request.PlanId && x.IsActive, ct);
   var ongoing = await MembershipHelper.GetAnyOngoingCardAsync(_db, visitorId, ct);
   if (ongoing is not null) return Conflict(new { message = "..." });

   var card = new MembershipCard { VisitorId = visitorId, PlanId = plan.Id, ... };
   _db.MembershipCards.Add(card);
   await _db.SaveChangesAsync(ct);
   return Ok(new { cardId = card.Id, ... });
}
```

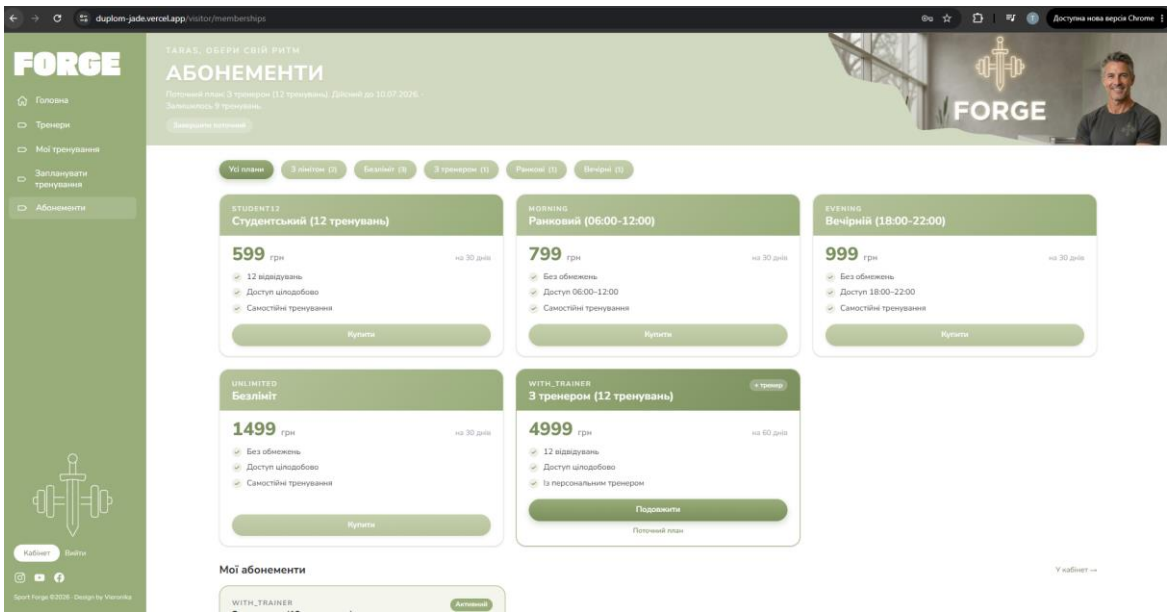

Рис. 3.5 Сторінка «Абонементи» з активним тарифом

Джерело: створено автором

Впровадження системи контролю доступу за ролями

Розмежування доступу базується на ролях Visitor, Trainer, Admin, що зберігаються в полі User.Role і дублюються в JWT-claims.

На сервері застосовуються атрибути:

```
[Authorize(Roles = "Admin")]
[Authorize(Roles = "Visitor")]
[Authorize(Roles = "Admin,Trainer")]
```

У Program.cs налаштовано AddAuthentication(JwtBearer) та AddAuthorization; ключ підпису, issuer і audience беруться з конфігурації Jwt (у production — змінні середовища Render).

Лістинг 3.5. Формування JWT із роллю користувача.

```
var claims = new List<Claim>
{
   new(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
   new(ClaimTypes.Role, user.Role.ToString()),
   new("role", user.Role.ToString()),
   ...
```

};

На клієнті компонент ProtectedRoute перевіряє наявність токена та відповідність ролі перед відображенням маршрутів /visitor/*, /trainer/*, /admin/*. Неавторизований користувач перенаправляється на /login.
Для адміністратора додатково передбачено поле AdminPermissions (бітова маска) для подальшого розширення прав без зміни ролі.

Бронювання сесій і заявки на індивідуальне тренування
Замість «улюблених товарів» у ForgeGym реалізовано облік бронювань (Booking) та заявок (TrainingRequest).

Сценарій	API	Результат
Групове заняття	POST .../sessions/{id}/book	Запис у bookings, списання відвідування
Індивідуальне	POST .../training-requests	Заявка зі статусом Pending, списання відвідування
Самостійне	POST .../self-bookings	Нова сесія + бронювання
Скасування	POST .../bookings/{id}/cancel	Статус Cancelled, повернення відвідування

Тренер переглядає відкриті заявки (GET .../requests/open), приймає або відхиляє їх; при прийнятті створюються TrainingSession і Booking, статус заявки — Accepted.

Відвідувач переглядає історію на сторінках «Мої тренування» та «Мої заявки» (MyTrainingsPage, MyRequestsPage).

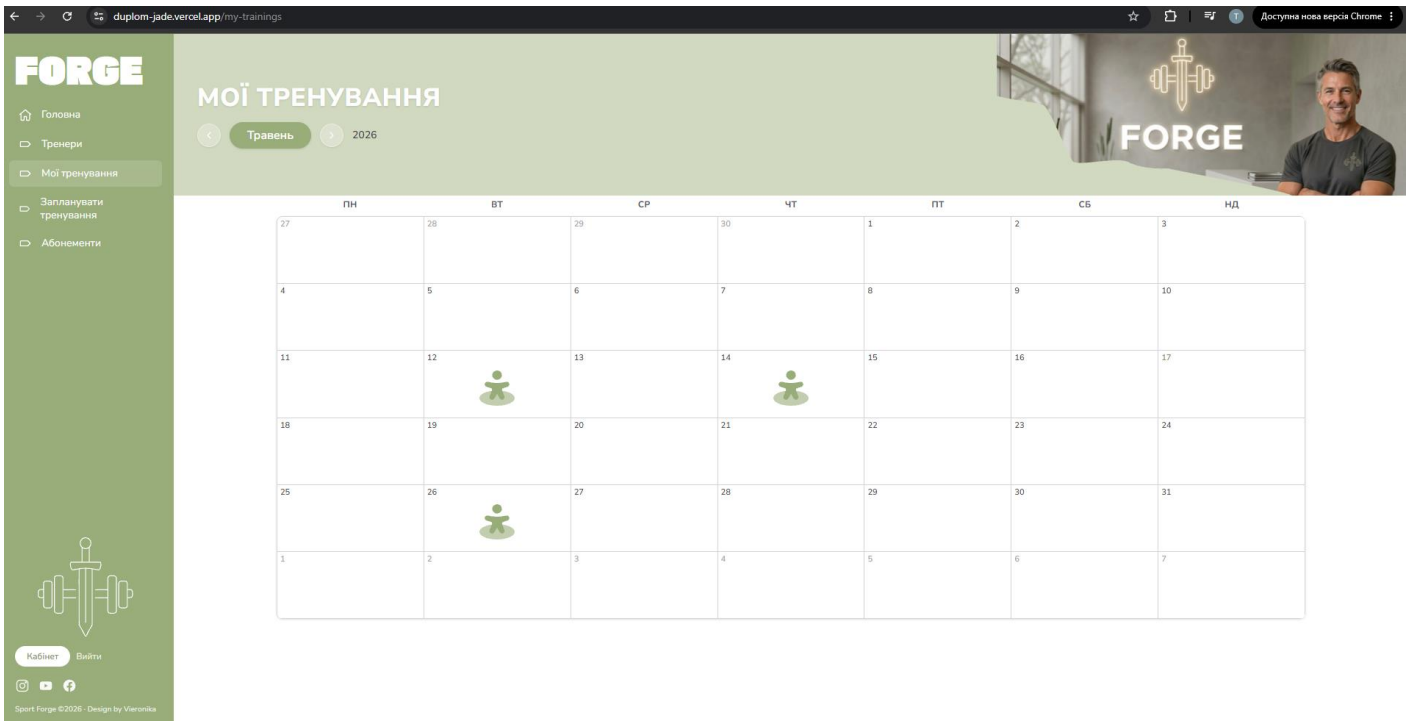

Рис. 3.6 Список майбутніх тренувань відвідувача

Джерело: створено автором

Система сповіщень користувача

Після зміни статусу заявки (прийняття/відмова) у таблицю notifications додається запис із типом Kind, заголовком Title, текстом Body та посиланням RelatedUrl (наприклад, /visitor/requests).

API сповіщень (NotificationsController):

- GET /api/notifications — список із лічильником непрочитаних;
- POST /api/notifications/{id}/read — позначити прочитаним;
- POST /api/notifications/read-all — прочитати всі.

На клієнті компонент NotificationsBell періодично оновлює кількість непрочитаних і відображає випадаючий список.

Лістинг 3.6. Створення сповіщення при прийнятті заявки тренером.

```
_db.Notifications.Add(new Notification
{
    UserId = entity.VisitorId,
    Kind = NotificationKind.TrainingRequestAccepted,
    Title = "Запит на тренування прийнято",
```

```
    Body = $"{trainer.FirstName} {trainer.LastName} прийняв ваш запит...",
    RelatedUrl = "/visitor/requests"
});
```

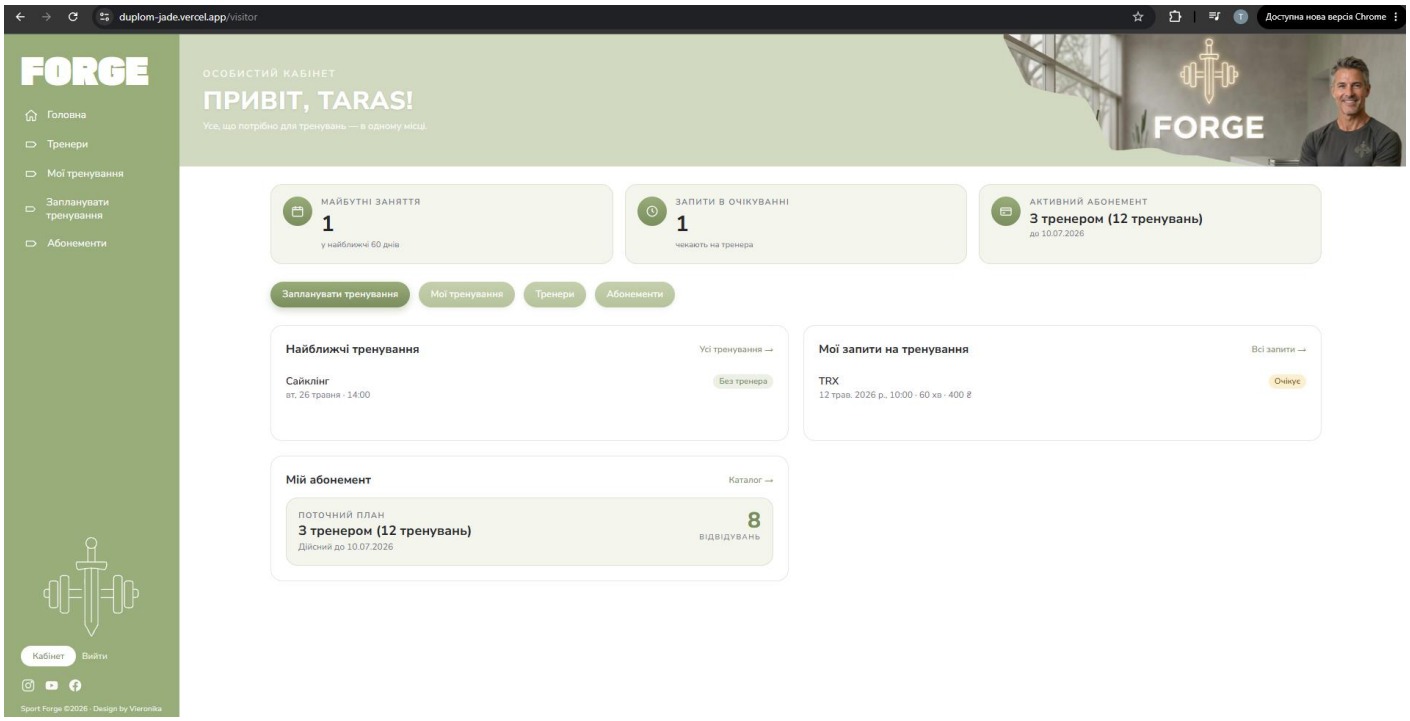

Рис. 3.7 Вигляд сповіщень у шапці застосунку

Джерело: створено автором

Розгортання та перевірка працездатності системи

Для промислового середовища налаштовано:

- Render Web Service — контейнер із ForgeGym.Api/Dockerfile, health-check /health, змінна DATABASE_URL для PostgreSQL;
- Render PostgreSQL — регіон Frankfurt, база forge_gym;
- Vercel — статична збірка React (npm run build), змінна VITE_API_URL вказує на URL API.

При старті API виконується DatabaseSeeder — створення облікового запису адміністратора з конфігурації SeedAdmin (лише якщо користувачів ще немає). CORS дозволяє запити лише з домену фронтенду, заданого в Cors:AllowedOrigins.

Модульні тести (ForgeGym.Tests) перевіряють валідатори вхідних даних (InputValidators — email, телефон, ім’я) за допомогою xUnit та FluentAssertions, що підвищує надійність API реєстрації та профілів

Висновки до розділу 3

У третьому розділі описано практичну реалізацію вебзастосунку ForgeGym для управління фітнес-клубом: від вибору технологічного стеку до розгортання в хмарі.

Обґрунтовано використання ASP.NET Core, Entity Framework Core та PostgreSQL, що забезпечило цілісну серверну архітектуру з міграціями та REST API. Реалізовано повний набір CRUD-операцій для адміністрування категорій, тарифів і розкладу.

Для відвідувачів створено зручний сценарій планування тренувань із фільтрацією за категорією, форматом заняття та спеціалізаціями тренерів. Впроваджено бізнес-правила обліку абонементів, рольовий контроль доступу на основі JWT, механізми бронювання та заявок, а також сповіщення про рішення тренера.

Система розгорнута на Render і Vercel, що підтверджує готовність рішення до експлуатації поза локальним середовищем розробки. Отриманий застосунок поєднує сучасні підходи до веб-розробки, автоматизацію процесів клубу та зручний інтерфейс для відвідувачів, тренерів і адміністрації.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було реалізовано повнофункціональний інтернет-магазин косметичних засобів, орієнтований на сучасні потреби користувачів, включно з підтримкою персоналізованого підбору продукції, голосової навігації та інтелектуальної чат-підтримки. Створений застосунок поєднує в собі широкий набір функцій, інтуїтивно зрозумілий інтерфейс та потужну серверну архітектуру, що дозволяє ефективно автоматизувати процес підбору та замовлення товарів.

На етапі дослідження проведено аналіз предметного середовища та існуючих інтернет-магазинів у сфері косметики. Вивчено функціональні можливості провідних платформ та виявлено їхні основні недоліки, такі як обмежений пошуковий функціонал, відсутність персоналізації та голосової взаємодії. Отримані результати обґрунтовують необхідність розробки нової онлайн-платформи, що враховує сучасні потреби користувачів і забезпечує розширені фільтри, індивідуальні рекомендації та інтеграцію голосового асистента.

В основі системи лежить сучасний стек технологій, який охоплює ASP.NET Core для побудови серверної логіки, Entity Framework Core для роботи з базою даних, PostgreSQL як надійну СУБД, JavaScript для реалізації клієнтської логіки. Завдяки цьому підходу вдалося досягти високого рівня функціональності, продуктивності та масштабованості, що є важливими критеріями для сучасних вебсистем.

Проведено моделювання бази даних з урахуванням усіх ключових сутностей та визначено основні зв’язки між таблицями. Розроблено функціональний вебінтерфейс, в якому користувач може здійснювати реєстрацію, переглядати каталог, фільтрувати продукти за численними критеріями, додавати товари в обране, формувати замовлення. Особливу увагу приділено фільтрації за складом, що дозволяє користувачеві виключати небажані інгредієнти. Такий функціонал є особливо важливим для осіб із алергіями або чутливою шкірою.

Інноваційною частиною проєкту є впровадження голосового асистента та інтелектуального чат-бота. Голосовий помічник реалізовано на базі Web Speech API, що дає змогу керувати інтерфейсом сайту, здійснювати пошук та отримувати поради без необхідності використання миші або клавіатури, що значно підвищує інклюзивність. Для реалізації AI-чат-підтримки застосовано локальну мовну модель Mistral у поєднанні з платформою Ollama, що забезпечує приватність даних і швидку обробку запитів. Також впроваджено фоновий сервіс для періодичного очищення історії чатів і автоматичного навчання AI-моделі на основі запитів користувачів. Це дозволяє підтримувати актуальність рекомендацій та забезпечити високий рівень адаптивності системи.

Розроблена система також включає механізм персоналізованих рекомендацій, які формуються на основі індивідуальних параметрів користувача. Це дозволяє запропонувати користувачу саме ті товари, які найбільше відповідають його потребам, підвищуючи задоволеність сервісом і ймовірність покупки. Важливим є те, що користувачі мають можливість виключати небажані інгредієнти з підбору товарів, що є ключовою функцією для сучасного косметичного маркетплейсу.

Загалом, виконана кваліфікаційна робота демонструє не лише практичне втілення теоретичних знань з програмування, баз даних, веброзробки та штучного інтелекту, але й уміння створити цілісну, сучасну, масштабовану та доступну інформаційну систему. Отриманий результат може слугувати основою для реального комерційного продукту, а також бути розширеним у напрямку мобільної версії, інтеграції з платіжними системами, багатомовної підтримки, машинного зору для аналізу шкіри тощо.

Отже, поставлені завдання було повністю реалізовано, а результати свідчать про доцільність та ефективність застосованих рішень. Запропонована система не лише вирішує базові завдання онлайн-магазину, але й виводить користувацький досвід на новий рівень, поєднуючи комфорт, безпеку та інтелектуальну допомогу у виборі косметичних засобів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Економічна правда. Цифровізація сфери послуг в Україні. URL: https://epravda.com.ua/news/2020/09/19/665297/ (дата звернення: 12.03.2026).
2. Sigma Software University. Що таке база даних? URL: https://university.sigma.software/what-is-database/ (дата звернення: 14.03.2026).
3. Microsoft. ASP.NET Core overview. URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core (дата звернення: 15.03.2026).
4. Microsoft. Tutorial: Create a minimal API with ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api (дата звернення: 17.03.2026).
5. Microsoft. Routing in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing (дата звернення: 18.03.2026).
6. Microsoft. Dependency injection in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection (дата звернення: 19.03.2026).
7. Microsoft. Configuration in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/ (дата звернення: 20.03.2026).
8. Microsoft. Загальні відомості про EF Core. URL: https://learn.microsoft.com/uk-ua/training/modules/persist-data-ef-core/2-understanding-ef-core (дата звернення: 21.03.2026).
9. Microsoft. Entity Framework Core Documentation. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 22.03.2026).
10. Microsoft. Creating and Configuring a Model. URL: https://learn.microsoft.com/en-us/ef/core/modeling/ (дата звернення: 24.03.2026).
11. Microsoft. EF Core Migrations Overview. URL: https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/ (дата звернення: 25.03.2026).
12. Microsoft. EF Core – Eager Loading with Include.

URL: https://learn.microsoft.com/en-us/ef/core/querying/related-data/eager (дата звернення: 26.03.2026).
13. Microsoft. EF Core – Basic Saving. URL: https://learn.microsoft.com/uk-ua/ef/core/saving/basic (дата звернення: 27.03.2026).
14. Somers P. Practical Entity Framework Core. Apress, 2022.
15. Shah R. Building APIs with ASP.NET Core. Packt Publishing, 2023.
16. Foxminded. Що таке PostgreSQL і для чого використовується? URL: https://foxminded.ua/postgresql-shcho-tse/ (дата звернення: 29.03.2026).
17. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 30.03.2026).
18. Npgsql. .NET Access to PostgreSQL. URL: https://www.npgsql.org/ (дата звернення: 31.03.2026).
19. Microsoft. Посібник зі зв’язків між таблицями. URL: https://support.microsoft.com/uk-ua/topic/посібник-зі-зв-язків-між-таблицями-30446197-4fbe-457b-b992-2f6fb812b58f (дата звернення: 02.04.2026).
20. Microsoft. LINQ Tutorial: Language Integrated Query. URL: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq (дата звернення: 03.04.2026).
21. Microsoft. Where Method (System.Linq.Enumerable). URL: https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.where (дата звернення: 05.04.2026).
22. Microsoft. Any Method (System.Linq.Enumerable). URL: https://learn.microsoft.com/uk-ua/dotnet/api/system.linq.enumerable.any (дата звернення: 06.04.2026).
23. Microsoft. Tutorial: Implement CRUD Functionality – ASP.NET Core MVC with EF Core. URL: https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/crud (дата звернення: 08.04.2026).
24. Thomas J. Securing ASP.NET Core Web Applications. Apress, 2021.
25. Microsoft. ASP.NET Core Authentication overview.

URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/ (дата звернення: 10.04.2026).
26. Microsoft. ASP.NET Core Authorization. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/introduction (дата звернення: 11.04.2026).
27. Microsoft. Simple authorization in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/simple (дата звернення: 12.04.2026).
28. Internet Engineering Task Force (IETF). RFC 7519: JSON Web Token (JWT). URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 14.04.2026).
29. Auth0. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction (дата звернення: 15.04.2026).
30. Microsoft. Authentication and authorization in minimal APIs. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis/security (дата звернення: 16.04.2026).
31. Microsoft. ASP.NET Core Web API documentation with Swagger / OpenAPI. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger (дата звернення: 18.04.2026).
32. Microsoft. Enable Cross-Origin Requests (CORS) in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/security/cors (дата звернення: 19.04.2026).
33. Microsoft. ILogger and Logging in C# / .NET. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/logging (дата звернення: 21.04.2026).
34. Microsoft. HttpClient Class. URL: https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient (дата звернення: 22.04.2026).
35. React. Офіційна документація бібліотеки React. URL: https://react.dev/ (дата звернення: 24.04.2026).
36. Vite. Next Generation Frontend Tooling. URL: https://vitejs.dev/ (дата звернення: 25.04.2026).
37. TypeScript. Офіційна документація мови TypeScript. URL: https://www.typescriptlang.org/docs/ (дата звернення: 27.04.2026).

38. React Router. Офіційна документація. URL: https://reactrouter.com/ (дата звернення: 28.04.2026).
39. Mozilla Developer Network. Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 29.04.2026).
40. Mozilla Developer Network. Window.localStorage. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (дата звернення: 30.04.2026).
41. Docker Inc. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 02.05.2026).
42. Microsoft. .NET Docker Samples and Best Practices. URL: https://learn.microsoft.com/en-us/dotnet/core/docker/ (дата звернення: 04.05.2026).
43. Render. Deploying a Docker service on Render. URL: https://render.com/docs/docker (дата звернення: 05.05.2026).
44. Render. Managed PostgreSQL Documentation. URL: https://render.com/docs/databases (дата звернення: 06.05.2026).
45. Vercel. Building and Deploying Vite Applications. URL: https://vercel.com/docs/frameworks/vite (дата звернення: 07.05.2026).
46. Vercel. Project Configuration with vercel.json. URL: https://vercel.com/docs/project-configuration (дата звернення: 08.05.2026).
47. Git SCM. Git Documentation. URL: https://git-scm.com/doc (дата звернення: 10.05.2026).
48. GitHub Docs. About GitHub Actions. URL: https://docs.github.com/en/actions (дата звернення: 11.05.2026).